

Laboratorio de Sistemas Electrónicos Digitales

Departamento de Ingeniería Electrónica
E.T.S.I. de Telecomunicación
Universidad Politécnica de Madrid

Memoria del proyecto desarrollado en el Laboratorio de Sistemas Electrónicos Digitales (LSED)

Curso 2007/2008

Título del proyecto desarrollado “Juego de “tres en raya” con efectos musicales basado en el MCF5272”

Autores (orden alfabético): Badolato Martín, Alejandro

Núñez Serrano, Fernando

Código de la pareja: MM-4

ÍNDICE GENERAL

1 INTRODUCCIÓN 2

2 DIAGRAMA DE SUBSISTEMAS 3

3 DESCRIPCIÓN DEL SUBSISTEMA HARDWARE 4

3.1 FILTRO RC PASO ALTO DE CONTINUA. 4
3.2 FILTRO PASO-BAJO DE ARMÓNICOS DE ALTA FRECUENCIA 5
3.3 AMPLIFICADOR DE POTENCIA. 9
3.4 FILTRADO DE LA ALIMENTACIÓN. 9

4 DESCRIPCIÓN DEL SUBSISTEMA SOFTWARE 11

4.1 PROCESO DEL PROGRAMA PRINCIPAL. 11
4.2 PROCESO DE LA INTERRUPCIÓN TIMER 1. 28
4.3 PROCESO DE LA INTERRUPCIÓN FEC. 29

5 DESCRIPCIÓN DE LAS MEJORAS 30

5.1 MEJORA DEL ALGORITMO DE JUEGO DE LA MÁQUINA. 30
5.2 DESARROLLO DEL SUBSISTEMA HARDWARE SOBRE PCB 32
5.3 DESARROLLO DE UNA APLICACIÓN SOBRE PC PARA DOTAR AL SISTEMA DE ENTORNO GRÁFICO. 33
5.4 USO DEL MÓDULO ETHERNET E IMPLEMENTACIÓN DE UNA LIBRERÍA UDP. 34
5.5 IMPLEMENTACIÓN GRÁFICA DE ESTADÍSTICAS DE JUEGO. 35
5.6 DESARROLLO DE UN TERMINAL UDP EN PC PARA PROBAR LA FUNCIONALIDAD DE LOS SISTEMAS (PLATAFORMA ENT Y APLICACIÓN PC) 36
5.7 DESARROLLO DE PROTOCOLO DE COMUNICACIONES PARA OBTENER VARIABLES DE PROGRAMA Y BLOQUES DE MEMORIA EN TIEMPO DE EJECUCIÓN MEDIANTE MÓDULO ETHERNET. 37

6 PRINCIPALES PROBLEMAS ENCONTRADOS 39

7 MANUAL DE USUARIO 40

8 BIBLIOGRAFÍA 41

9 ANEXO I: CÓDIGO DEL PROGRAMA DE LA PRIMERA SESIÓN 42

10 ANEXO II: CÓDIGO DEL PROGRAMA DEL PROYECTO FINAL 48

10.1 CODIGO VISUAL BASIC APLICACIÓN 3 EN RAYA 48
10.2 CODIGO VISUAL BASIC APLICACIÓN TERMINAL UDP 62
10.3 CODIGO APLICACIÓN 3 EN RAYA (COLDFIRE) 3RAYA.LIB 64
10.4 CODIGO APLICACIÓN 3 EN RAYA (COLDFIRE) 3RAYA.LIB 110

1 Introducción

El objetivo de la práctica a nuestro modo de entender es familiarizarse con un entorno de trabajo basado en el microcontrolador estudiado en la asignatura SEDG, enfrentarse a los problemas reales en la programación de microcontroladores y solucionar de una manera eficiente los problemas planteados para cumplir con los objetivos marcados en el plazo fijado.

El objetivo de la práctica era el de desarrollar un sistema que fuera capaz de jugar al tres en raya con dos modalidades de juego: Contra la máquina y modo 2 jugadores. En este apartado lo más difícil ha sido implementar un algoritmo eficiente para la jugada automática por parte de la máquina. Además se pedía una implementación musical mediante el uso de temporizadores y una adaptación de la señal de audio mediante circuito hardware.

Estos objetivos han sido superados en un breve plazo de tiempo, lo que nos ha permitido desarrollar mejoras. Estas han sido las siguientes:

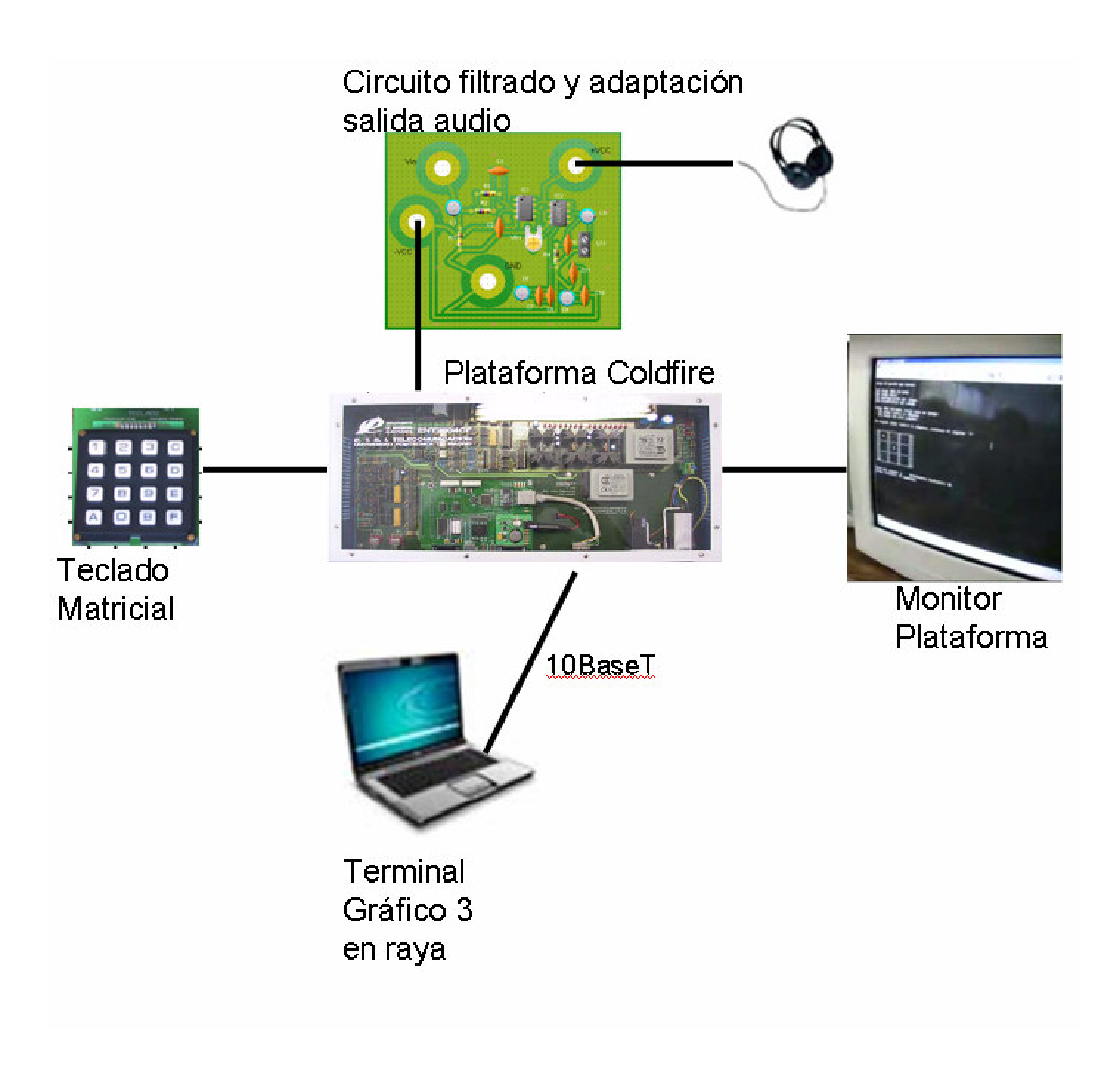
- Desarrollar una aplicación en PC para dotar al sistema de entorno gráfico.
- Implementar una comunicación UDP entre aplicación PC y plataforma.
- Mejorar el algoritmo de juego propuesto de la máquina.
- Realizar el hardware sobre circuito impreso PCB.
- Desarrollar un entorno de debugging desde la aplicación del PC que permite lectura de variables en tiempo de ejecución y volcados de memoria.
- Implementar un sistema gráfico de estadísticas.
- Desarrollo de un terminal de comunicaciones UDP en PC para probar la funcionalidad de los sistemas (plataforma ENT y aplicación PC)
- Desarrollo de protocolo de comunicaciones para obtener variables de programa y bloques de memoria en tiempo de ejecución mediante módulo ethernet.

Tenemos una idea aproximada de la dificultad que conlleva el desarrollo de soluciones a nivel de microcontrolador y podemos ser capaces de analizar las limitaciones de desarrollo posible para este tipo de plataformas.

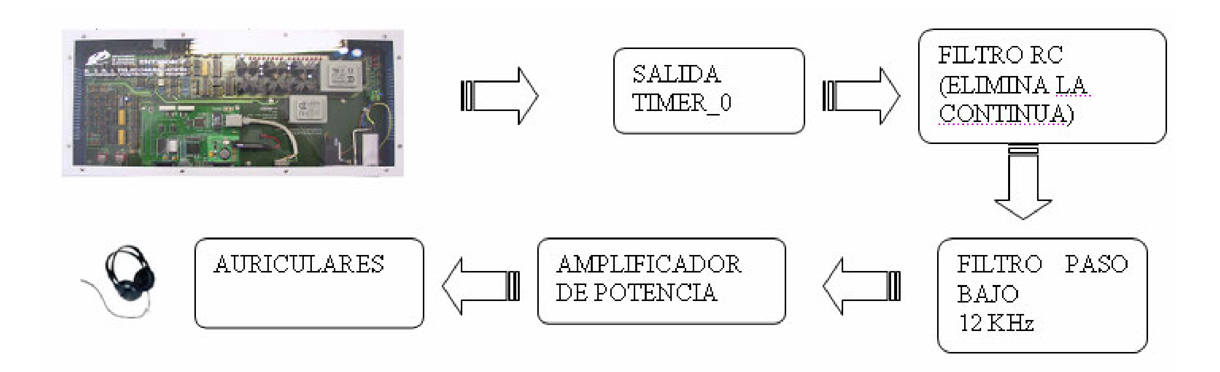
Después de realizar esta práctica hemos aprendido bastante sobre el lenguaje de programación C y nos hemos familiarizado con los entornos de desarrollo de microcontroladores.

A lo largo de la siguiente memoria encontrarán información más detallada sobre el trabajo desarrollado.

2 Diagrama de subsistemas



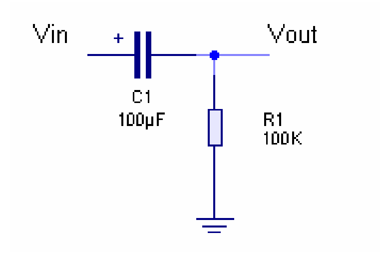
3 Descripción del subsistema Hardware



El hardware utilizado en este sistema tiene el objetivo de adaptar la salida de la placa entrenadora de ColdFire a cualquier auricular externo. Para ello, se realiza en primer lugar una etapa de filtrado en la que se elimina la continua mediante un filtro RC y se elimina los armónicos de alta frecuencia innecesarios en nuestro sistema. A continuación se aplica una etapa de amplificación tras una previa atenuación que nos permita obtener una señal de potencia suficiente sin distorsión alguna. Los módulos que intervienen serán detallados a continuación.

3.1 Filtro RC paso alto de continua.

Hemos realizado un filtro paso alto mediante la inclusión de una red RC. Si analizamos el circuito por mayas obtenemos operando que la frecuencia de corte es: $f = 1 / 2*\pi*R*C$. Por lo tanto, como nuestro objetivo es eliminar la continua, debemos hacer que esta sea lo menor posible para evitar afectar a mi señal. Debido a esto, escogimos los valores R=100K y C=100uF, obteniendo una frecuencia de corte de f=0,016Hz. A continuación mostramos el circuito equivalente:



3.2 Filtro Paso-Bajo de armónicos de alta frecuencia

En nuestra placa hemos empleado un filtro Paso-Bajo de Sallen-Key de segundo orden para eliminar los armónicos en alta frecuencia generados por la onda cuadrada. Por consiguiente el esquema de nuestro filtro es el siguiente:

Según las especificaciones del enunciado de la práctica de LCEL. Calculamos la función de transferencia:

$$A_V = \frac{A_{vm}}{\frac{s^2}{\omega_0^2} + \frac{s}{\omega_0 Q} + 1}$$

Si escribimos la fórmula genérica de la función de transferencia de dos polos (en términos simples), operamos e identificamos con la expresión anterior, comprobamos que el circuito tiene 2 polos justo en la frecuencia de corte. Además también de esta expresión podemos obtener tanto el valor de Q como el de f_0.

$$f_0 = \frac{1}{2\pi\sqrt{mn}RC} \qquad Q = \frac{\sqrt{mn}}{m+1}$$

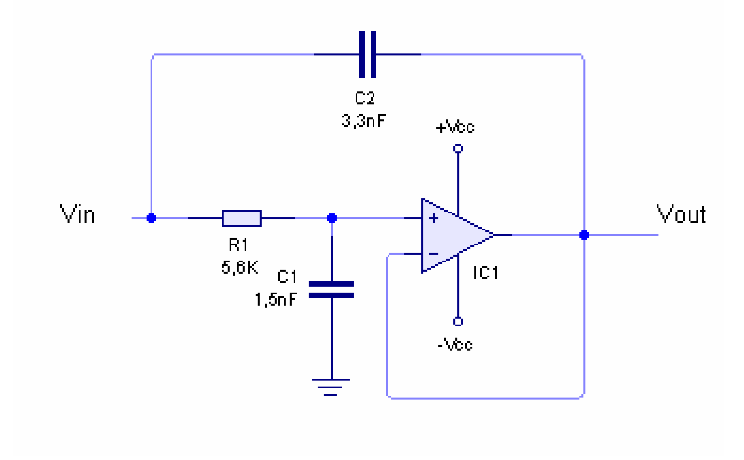
Siguiendo las indicaciones del enunciado calculamos los valores para el comportamiento deseado:

Microsoft Excel - Copia de SALLENKEY12KHz.xls

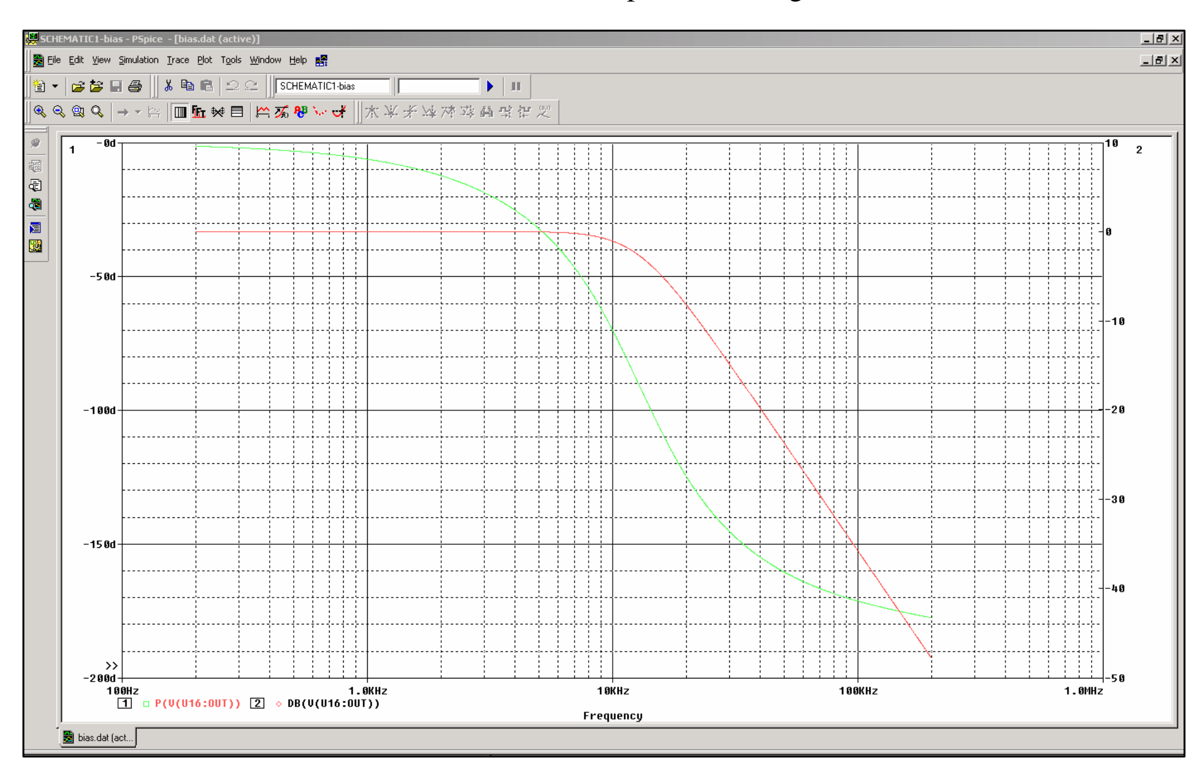
D7 | Adim

	A	B	C	D	E	F	G	H	I	J
1										
2			FILTRO SALLEN KEY							
3										
4										
5		R*	56000	ohms						
6		C*	1,6747E-10	F	0,00016747	uF	0,167469553	pF	167,469553	nF
7		Q	0,707106781	Adim						
8		f0*	12000	Hz						
9										
10		n*	2							
11		nC*	3,34939E-10	F	0,00033494	uF	0,334939106	pF	334,939106	nF
12										
13		C	1,5E-09	F	0,0015	uF	1,5	pF	1500	nF
14		K	2							
15		m	1							
16		R	6252,19664							
17		f0*	1339,756423							
18										

En el siguiente esquema mostramos el circuito resultante con los datos obtenidos.



A continuación realizamos una simulación en Pspice con los siguientes resultados:



Una vez diseñado y montado el filtro realizamos las medidas en el laboratorio. La forma de trabajar es la que aprendimos LCEL. Aislamos el filtro del resto de elementos circuitales. Conectamos a la entrada una señal sinoidal con una tensión Vinpp usando el generador de funciones y variamos su frecuencia según la medida a realizar. A la entrada conectamos una sonda del osciloscopio y a la salida otra. Ajustamos la frecuencia deseada y verificamos que el valor de la entrada no ha variado. Medimos la amplitud de salida.

$$Atenuacion = 20\log\left(\frac{V_{out}}{V_{in}}\right)dB$$

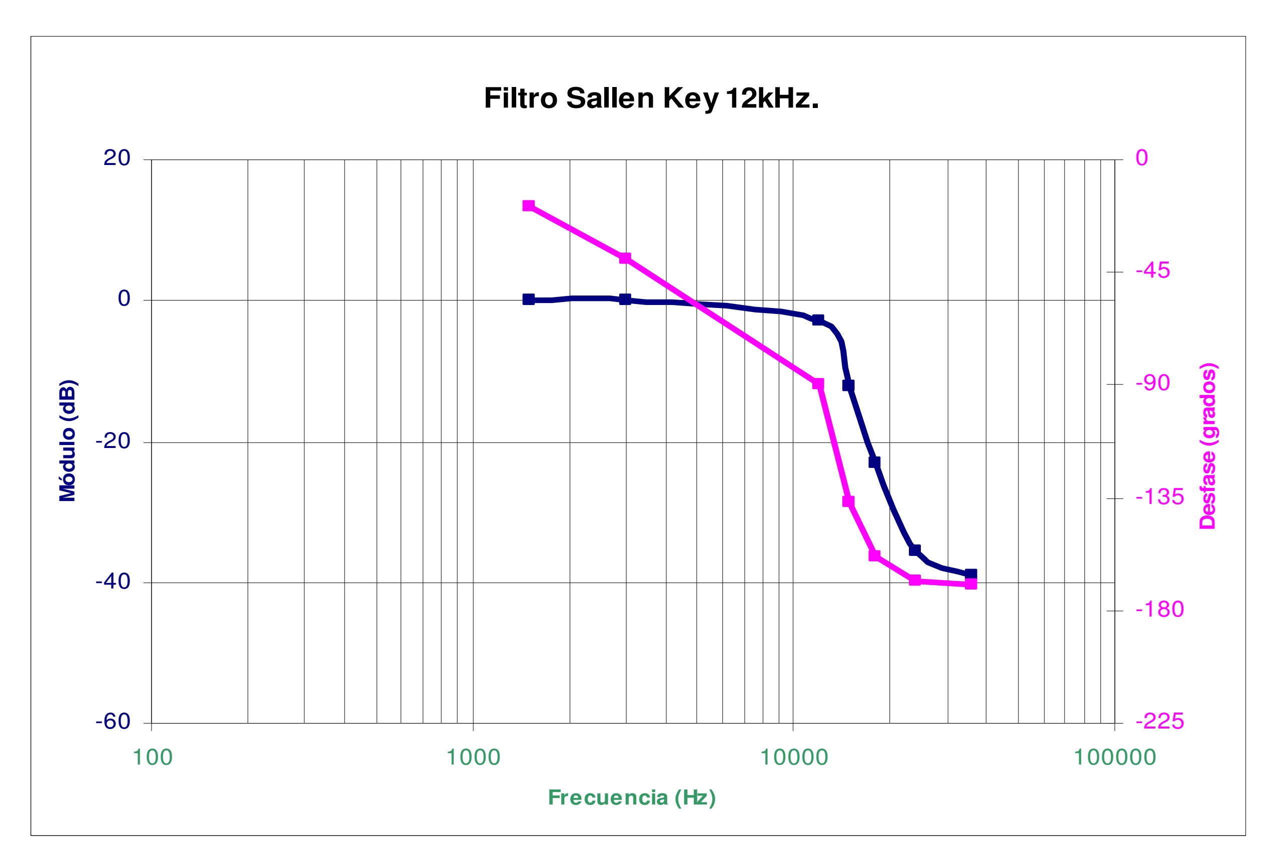
Para los cálculos de la fase hacemos uso del modo XY del osciloscopio para representar las figuras de Lissajous. Aparece una elipse. El corte con el eje y de la elipse centrada es la medida a, la proyección del máximo de la elipse sobre este mismo eje es la medida b. Mediante la fórmula

$$Fase = arcseno\left(\frac{b}{a}\right)rad$$

Para facilitarnos la tarea, simplificar los cálculos y disminuir la acumulación de errores el registro de las medidas se realiza directamente sobre Excel. Una vez finalizada la adquisición de datos generamos la gráfica correspondiente. Puesto que el 0 no existe en la escala logarítmica, aproximamos por 0,01 para poder representar mejor el diagrama de Bode.

Microsoft Excel - SALLENKEY12KHz.xls

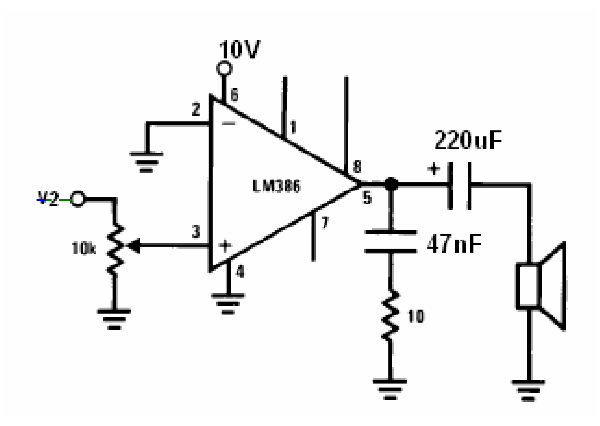
	C	D	E	F	G	H	I	J	K
13	frecuencia	1500	3000	12000	15000	18000	24000	36000	Herzios
14	Vinpp	2	2	2	2	2	2,8	2,73	voltios
15	Voutpp	2	2	1,45	0,489	0,139	0,0469	0,0304	voltios
16	Módulo dB	0	0	-2,79323987	-12,2344227	-23,1603039	-35,5197038	-39,0657813	dB
17	a	7	14	1	15	8	5	4	Adim
18	b	22	22	1	22	22	24	23	Adim
19	fase (rad)	0,32381102	0,689775	1,57079633	0,75024524	0,37216853	0,20987059	0,17480188	radianes
20	fase (grados)	18,5530045	39,5211964	90	42,9858861	21,3236863	12,0246992	10,0154102	grados
21	fase (grados)	18,5530045	39,5211964	90	137,014114	158,676314	167,975301	169,98459	grados
22	fase (grados)	-18,5530045	-39,5211964	-90	-137,014114	-158,676314	-167,975301	-169,98459	grados



3.3 Amplificador de Potencia.

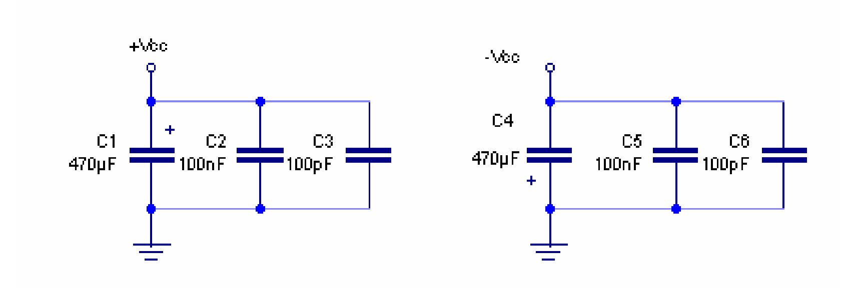
Para evitar saturar la salida del operacional del filtro anterior, hemos equipado al receptor con un amplificador de potencia que sea capaz de generar los niveles requeridos por los auriculares. Mediante un potenciómetro somos capaces de controlar la ganancia de dicho amplificador, aunque por tratarse de ganancia lineal y no logarítmica apenas es apreciable por el oído humano.

Para la construcción de este amplificador hemos utilizado el esquema propuesto por el fabricante con el integrado del LM386 con los valores expuestos en el siguiente diagrama:

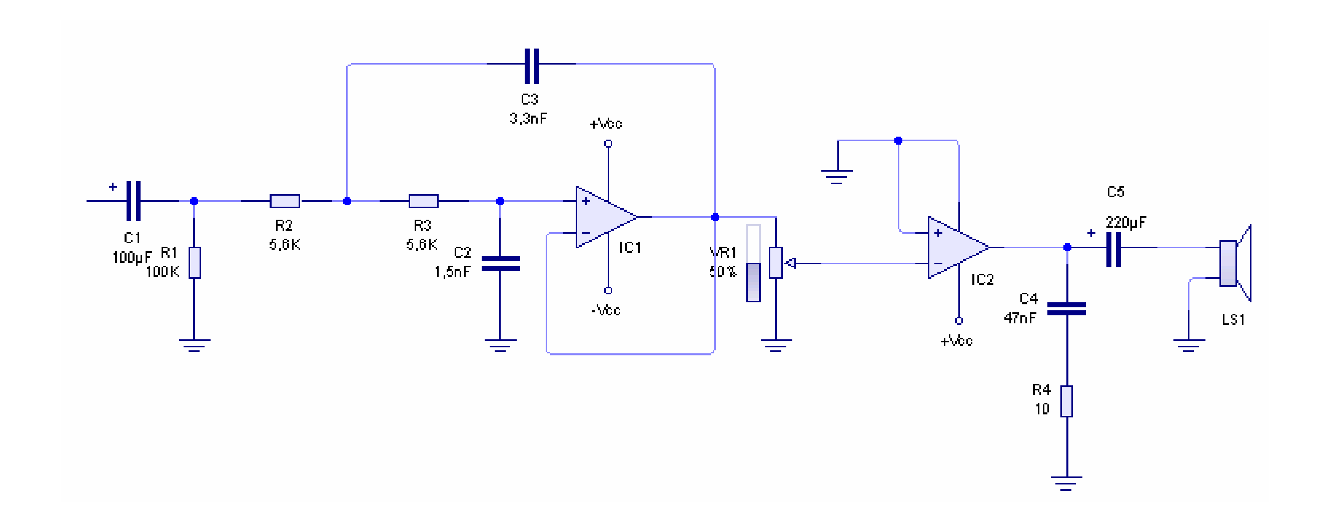


3.4 Filtrado de la alimentación.

Las alimentaciones han sido convenientemente filtradas para evitar perturbaciones en la alimentación de los circuitos. De este modo cada fuente:+10 y -10 Vcc dispone justo a su entrada de un conjunto de 3 condensadores de 100uF (electrolítico) 1uF (poliéster) y 5pF (cerámico) que filtran los ruidos que por sus líneas se pudieran inducir.

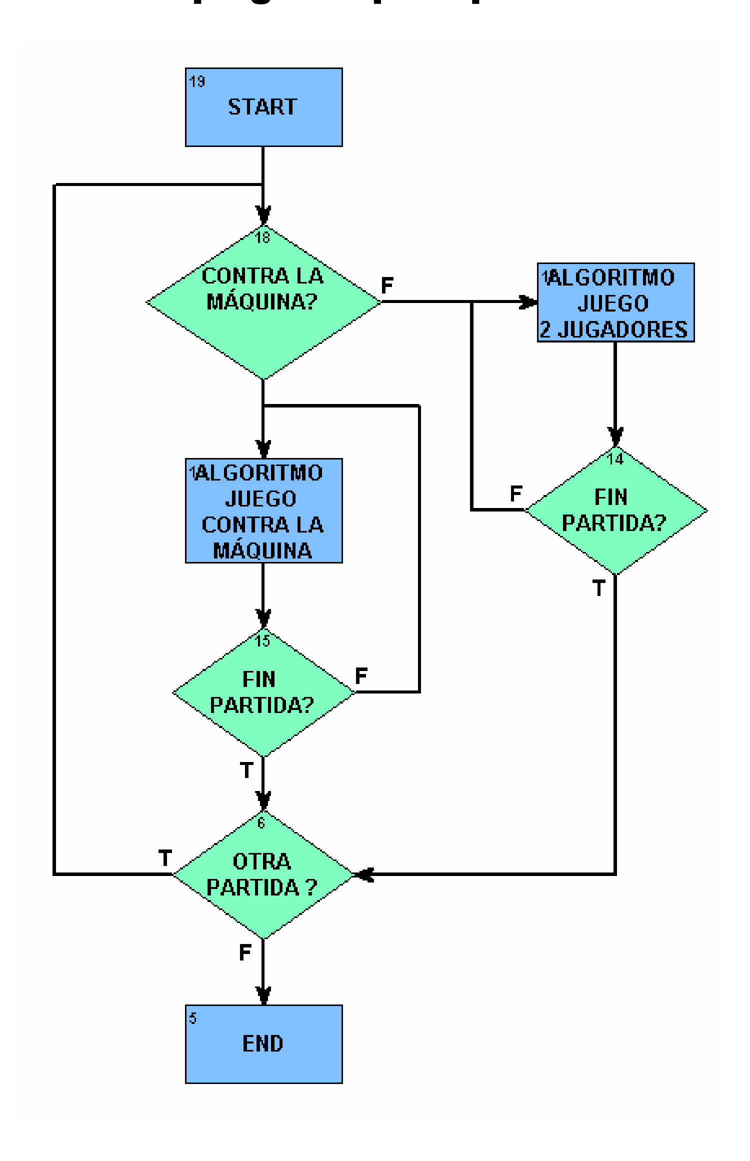


Finalmente el esquema completo del circuito queda de la siguiente manera:

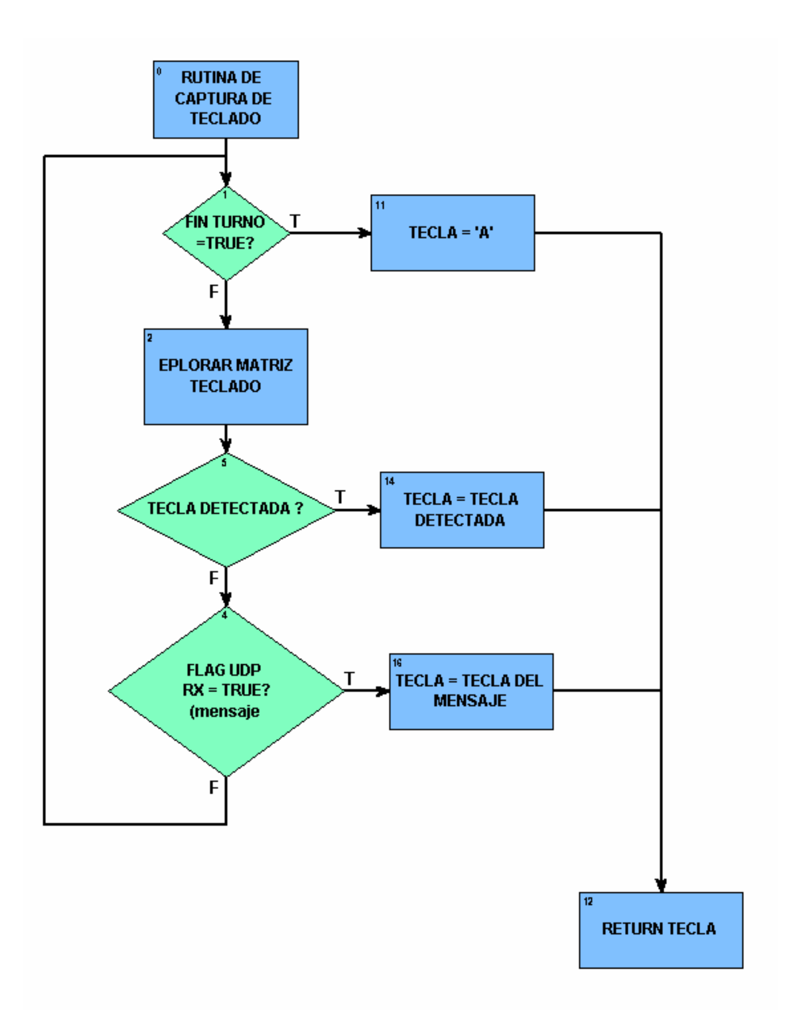


4 Descripción del subsistema Software

4.1 Proceso del programa principal.



1. Subrutina teclado( ).



2. Subrutina enDiagonalIzda( ).

Variables de Entrada: WORD fila, WORD columna.
Variables de Salida: BOOL temp.
Comprueba si la celda está en la diagonal izquierda, si es así retorna temp = TRUE.

3. Subrutina muestraTablero ( ).

Variables de Entrada: void.
Variables de Salida: void
Variables globales:
Muestra el tablero en pantalla y llama a muestraTableroUdp().

4. Subrutina muestraTableroUdp ( ).

Variables de Entrada: void.
Variables de Salida: void.
Envía el tablero por UDP poniendo antes el código de envío.

5. Subrutina iniciaTablero ( ).

Variables de Entrada: void.
Variables de Salida: void.
Variables globales: tablero [][], numeroFichas, numeroFichasMaquina.
Inicializa el tablero con VACIO = '_' .

6. Subrutina cambiaJugador ( ).

Variables de Entrada: void.
Variables de Salida: void.
Variables globales: turnoActual.
Cambia el turno actual del jugador.

7. Subrutina cambiaJugadorUdp ( ).

Variables de Entrada: void.
Variables de Salida: void.
Variables globales: turnoActual.
Cambia el turno actual del jugador actual y envía cambio al cliente udp poniendo antes el código de envío.Subrutina Subrutina enDiagonalIzda( ).

8. Subrutina símbolo ( ).

Variables de Entrada: WORD jugador.
Variables de Salida: BYTE FICHAS1 = 'X', BYTE FICHAS2 = 'O'.
Retorna ficha del jugador.

9. Subrutina esEsquina ( ).

Variables de Entrada: WORD fila, WORD columna.
Variables de Salida: BOOL.
Retorna TRUE si es la celda es esquina.

10. Subrutina contadorColumna ( ).

Variables de Entrada: WORD columna.
Variables de Salida: BYTE contador.
Variables globales: turnoActual.
Retorna el número de fichas del turnoActual en la columna, el código se basa en un bucle for que recorre el tablero y cuenta el número de fichas del jugador del turno actual:

```
   for(j=0;j<MAXFILAS;j++){
if (tablero[j][columna]==simbolo(turnoActual)){
contador++;
}
```

11. Subrutina contadorFila ( ).

Variables de Entrada: WORD fila.
Variables de Salida: WORD contador.
Variables globales: turnoActual.
Retorna nº de fichas del turnoActual en la fila de manera similar a contadorColumna().

12. Subrutina contadorDiagonalIzda ( ).

Variables de Entrada: void.
Variables de Salida: WORD contador.
Variables globales: turnoActual.
Retorna nº de fichas del turnoActual en la diagonal izquierda, para ello hacemos lo siguiente:

```
if (tablero[0][0]==simbolo(turnoActual)) contador++;
if (tablero[1][1]==simbolo(turnoActual)) contador++;
if (tablero[2][2]==simbolo(turnoActual)) contador++;
```

13. Subrutina contadorDiagonalDcha ( ).

Variables de Entrada: void.
Variables de Salida: WORD contador.
Variables globales: turnoActual.
Retorna el número de fichas del turnoActual en la diagonal derecha de manera similar a contadorDiagonalIzda.

14. Subrutina victoriaEnFila ( ).

Variables de Entrada: void.
Variables de Salida: BOOL.
Retorna TRUE si el turnoActual ha ganado por filas, para ello usamos un bucle for que recorre las tres filas comprobando si en alguna hay tres en raya.

```
for(i=0;i<MAXFILAS;i++){
     // contar las fichas en esa fila y ver si hay 3
    if (contadorFila(i)==3) return TRUE;
}
```

15. Subrutina victoriaEnColumna ( ).

Variables de Entrada: void.
Variables de Salida: BOOL.
Retorna TRUE si el turnoActual ha ganado por columnas de manera similar a victoriaEnFila.

16. Subrutina victoriaEnDiagonal ( ).

Variables de Entrada: void.
Variables de Salida: BOOL
Retorna TRUE si el turnoActual ha ganado por diagonales:

```
// comprobar diagonal IZDA
if (contadorDiagonalIzda()==3) return TRUE;
// comprobar diagonal DCHA
if (contadorDiagonalDcha()==3) return TRUE;
```

17. Subrutina compruebaVictoria ( ).

Variables de Entrada: void.
Variables de Salida: BOOL.
Retorna TRUE si el turnoActual ha ganado la partida (por fila o por columna o por diagonal) y envía por udp tiempo de turno restante (0).

18. Subrutina esCeldaCentral ( ).

Variables de Entrada: WORD fila, WORD columna.
Variables de Salida: BOOL
Retorna TRUE si es la celda central, es decir:

```
if (fila==1 && columna==1)
```

19. Subrutina esCeldaValida ( ).

Variables de Entrada: WORD fila, WORD columna.
Variables de Salida: BOOL.
Comprobar que la celda está en límites, retorna TRUE si es una celda del tablero, es decir:

```
if (fila>=0 && columna>=0 && fila<3 && columna<3)
```

20. Subrutina esCeldaVacia ( ).

Variables de Entrada: WORD fila, WORD columna.
Variables de Salida: BOOL.
Retorna TRUE si es una celda vacía, es decir:

```
if (tablero[fila][columna]==VACIO)
```

21. Subrutinas comprobarfila, comprobarColumna.

Variables de Entrada: WORD fila, WORD columna.
Variables de Salida: WORD importancia.
Estas dos subrutinas funcionan igual, retornan la importancia de la casilla según la fila o la columna respectivamente con el siguiente orden de prioridad: GANA, EVITARPERDER, SIPUEDEGANARESQUINA, SI PUEDEGANAR, NOPUEDEGANARESQUINA Y NOPUEDEGANAR. Para ello contamos en la fila o columna el número de de fichas mías y las del contrario (subrutinas contadorColumna y contadorFila) y vemos la importancia de la siguiente manera:

```
if ((mias==2) && (contrario==0)) return GANA;
     if ((contrario==2) && (mias==0)) return EVITAPERDER;
     if (contrario==0 && mias==1){
      if (esEsquina(fila,columna)==TRUE){
             return SIPUEDEGANARESQUINA;
        }else return SIPUEDEGANAR;
     }

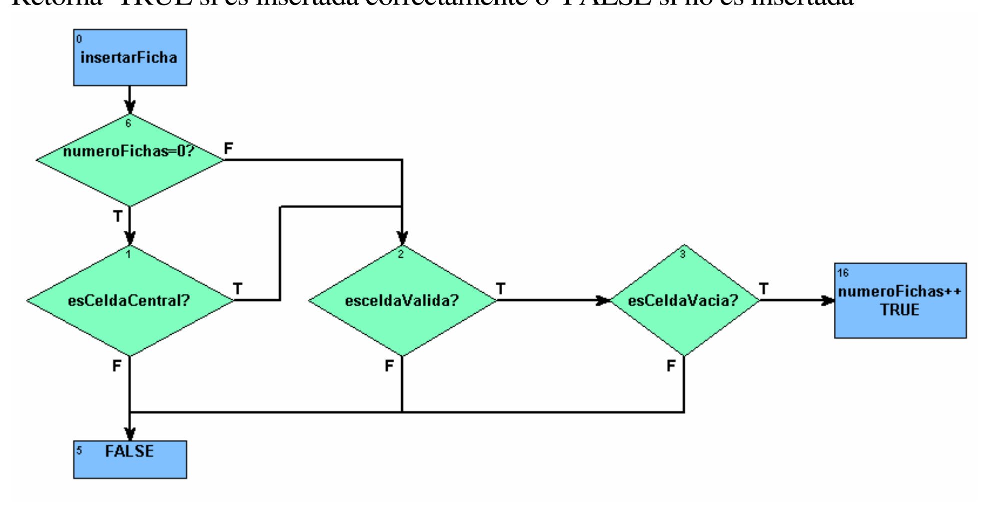
      // resto de los casos
     if (esEsquina(fila,columna)==TRUE){
         return NOPUEDEGANARESQUINA;
     }
```

22. Subrutinas comprobarDiagonalIzda, comprobarDiagonalDerecha.

Variables de Entrada: void.
Variables de Salida: WORD importancia.
Estas dos subrutinas funcionan igual que las anteriores pero en este caso no hace falta pasarle como parámetro la fila y la columna.

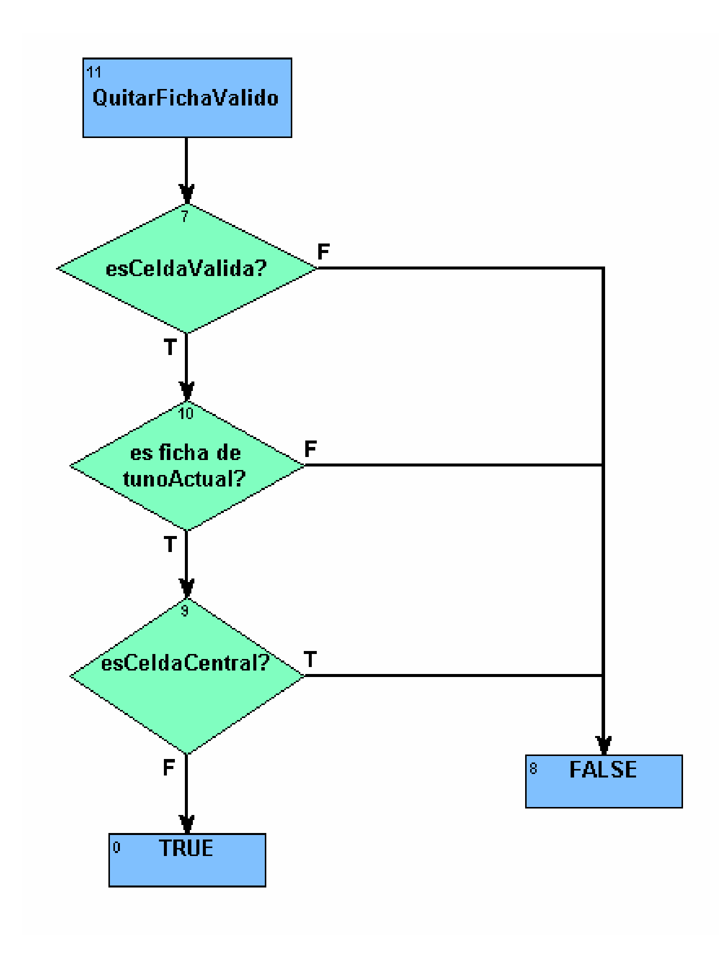
23. Subrutina insertarFicha.

Variables de Entrada: WORD fila, WORD columna.
Variables de Salida: BOOL.
Variables globales: turnoActual.
Retorna TRUE si es insertada correctamente o FALSE si no es insertada



24. Subrutina quitarFichaValido:

Variables de Entrada: WORD fila, WORD columna.
Variables de Salida: BOOL.
Variables globales: turnoActual.
Retorna TRUE si se puede quitar la ficha.



25. Subrutina quitarFicha:

Variables de Entrada: WORD fila, WORD columna.
Variables de Salida: BOOL.
Variables globales: numeroFichas.
Quita una ficha del tablero verificando que se quita una ficha del jugador que tiene el turno actual.
Retorna TRUE si es eliminada correctamente o FALSE si no es eliminada.

26. Subrutina selTipoJuego:

Variables de Entrada: BYTE tecla.
Variables de Salida: void
Variables globales: BYTE tecla, WORD tipoJuego.
Método que selecciona tipo de Juego.
Si tecla = 'A':

- tipoJuego=UNJUGADOR.
- Imprimimos por pantalla inicio juego contra la máquina.
- Enviamos por udp.

Si tecla = 'B':

- tipoJuego= DOSJUGADORES.
- Imprimimos por pantalla inicio juego entre dos jugadores.
- Enviamos por udp.

27. Subrutina leerCelda:

Variables de Entrada: void.
Variables de Salida: void
Variables Globales: numFila, numColumna, finTurno.
Captura una selección de celda.

- Bucle (do-while) hasta selección correcta de celda.
  - Hacemos un switch con 9 case, uno para cada tecla del tablero.
  - Si (!esCeldaValida(numFila,numColumna))&(finTurno==FALSE) la celda es incorrecta.
- Mientras la celda sea incorrecta y no halla acabado el tiempo del turno repetimos la operación.

28. Subrutina desplazar:

Variables de Entrada: void.
Variables de Salida: BOOL.
Variables globales: WORD numeroFichasMaquina.
Comprueba si tiene que desplazar o insertar ficha mirando si hay 3 fichas de la máquina en el tablero.

29. Subrutina jugadaJugador:

Variables de Entrada: void.
Variables de Salida: void.
Variables globales: BOOL finTurno.
Jugada de un Jugador:

- Hacemos copia de seguridad del tablero y del número de fichas en el tablero
- Hacemos un flag: *fallo* que nos avisa si ha habido algún error insertado o quitando alguna ficha.
- Bucle (do-while) hasta que se ejecute todo correctamente.
    - Si hay que desplazar ficha, pedimos selección de ficha a mover y llamamos a *leerCelda()* para capturar la selección.
    - Si no ha acabado el tiempo de nuestro turno y no ha habido error al quitar la ficha (*fallo= FALSE*).
        - Mostramos el tablero y pedimos donde poner la ficha.
        - Capturamos la selección y actualizamos *fallo* por si ha habido error al insertar en la celda.
    - Si no hay que desplazar ficha porque todavía no están las seis fichas en el tablero pedimos selección de ficha a mover y llamamos a *leerCelda()* para capturar la selección y actualizamos el valor de *fallo* por si ha habido error al insertar en la celda.
    - Si ha habido fallo porque hallamos insertado en una celda que no podemos.
        - La jugada es incorrecta y pedimos repetir la jugada.
        - Recuperamos el tablero y el número de fichas.
        - Mostramos el tablero.

- Repetimos esta operación mientras no halla acabado el tiempo de nuestro turno y halla habido error al quitar la ficha (*fallo= TRUE*).
- Si en algún momento se acaba el tiempo de nuestro turno (*finTurno==TRUE*) se sale del bucle.
    - Avisamos por pantalla que se ha expirado el tiempo.
    - Recuperamos el tablero y el número de fichas.

30. Subrutina jugadaMaquina:

Variables de Entrada: void.
Variables de Salida: void.
Jugada de la Maquina:

- Inicializamos el nivel de importancia.
- Si hay que desplazar ficha.
    - Llamamos a *jugadaAutomaticaDesplaza().*
- Si no hay que desplazar ficha.
    - Llamamos a *jugadaAutomatica().*

31. Subrutina jugarTurno:

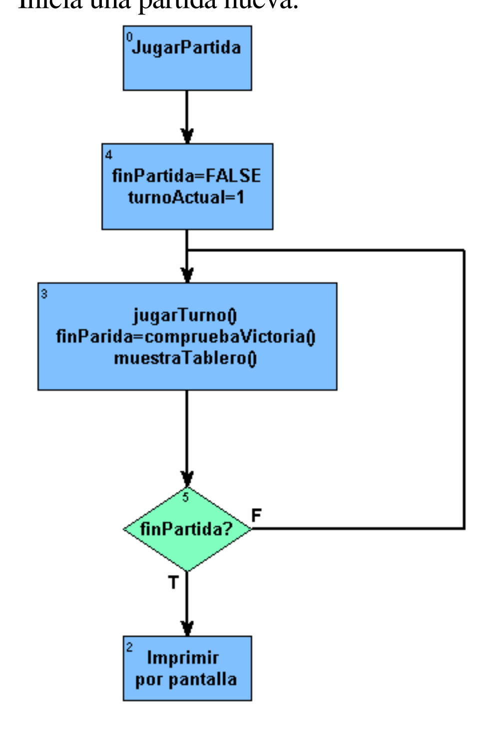
Variables de Entrada: void.
Variables de Salida: void.
Variables globales: WORD turnoActual, WORD tiempoTurno, WORD tipoJuego.
Juega un turno de turnoActual indicando por pantalla quien realiza el movimiento.

- Inicializamos tiempo del turno y lo enviamos por udp.
- Inicializamos flag: finTurno=FALSE.
- Hacemos un switch de la siguiente manera: (código comprimido)

```
switch(tipoJuego){
     case 1: // 1 jugador contra la maquina
          switch(turnoActual){
             case 1: // turno del jugador
                jugadaJugador();
             case 2: // Turno de la maquina
                jugadaMaquina();
          }
     case 2: //2 jugadores
            if (turnoActual==1){
                  output("\r\n Turno del jugador 1\r\n");
            }else output("\r\n Turno del jugador 2\r\n");
             jugadaJugador();
 }
```

32. Subrutina jugarPartida:

Variables de Entrada: void.
Variables de Salida: void.
Variables globales:
Inicia una partida nueva.



33. Subrutina mostrarMenu:

Variables de Entrada: void.
Variables de Entrada: void.
Variables globales: WORD tipoJuego.
Muestra el menú del juego: (código comprimido)

```
output("Elija el tipo de juego desdeado: (A) contra la máquina (B) entre
dos Jugadores:");
 // selecionar tipo juego
do{
   selTipoJuego(teclado());
   if (tipoJuego==0) {
      output("\r\n Selección incorrecta. Repita selección \r\n");
   }
}while(tipoJuego==0);
```

34. Subrutinas iniciaMelodia, iniciaMelodiaGanador, iniciaMelodiaPerdedor:

Variables de Entrada: void.
Variables de Salida: void.
Variables globales: WORD duracionTotalNotaActual, WORD duracionNotaActual, WORD duracionTotalNotaActual, BOOL suenaMelodia, BOOL suenaGanador, BOOL suenaPerdedor.
Con estas tres subrutinas se arranca el timer0 con la primera nota de la melodía de cada caso:

```
  ajustaTimer0(calculaPeriodo(melodia[notaActual]));
  ajustaTimer0(calculaPeriodo(melodiaGanador[notaActual]));
  ajustaTimer0(calculaPeriodo(melodiaPerdedor[notaActual]));
```

notaActual es un puntero que marca la frecuencia de la nota que se está reproduciendo.

35. Subrutina esMejorJugada:

Variables de Entrada: WORD mejorValor, WORD jugadaActual.
Variables de Salida: BOOL.
Devuelve TRUE si la jugadaActual tiene más puntuación que la almacenada hasta el momento.

```
  if (jugadaActual>=mejorValor){
          return TRUE;
  }
```

36. Subrutina buscaMejorOpcion:

Variables de Entrada: WORD fila, WORD columna.
Variables de Salida: WORD importanciaCasilla.
Devuelve la mayor importancia que tiene la celda. Se vigila la fila, la columna y si es el caso la diagonal asociada a esa posición. Según el siguiente criterio de más a menos: GANA, EVITARPERDER, SIPUEDEGANARESQUINA (en siguiente turno), SI PUEDEGANAR, NOPUEDEGANARESQUINA Y NOPUEDEGANAR. Lo mejor para entender está subrutina es ver el código:

```
  // inicializamos importancias:
     WORD importanciaCasilla=NOPUEDEGANAR; // de la casilla
     WORD importanciaF=NOPUEDEGANAR; // de su fila
     WORD importanciaC=NOPUEDEGANAR; // de su columna
     WORD diagonalIzda=NOPUEDEGANAR; // diagonal IZDA
     WORD diagonalDcha=NOPUEDEGANAR; // diagonal DCHA
     // calcular importancia de la fila
     importanciaF=comprobarFila(fila,columna);
     // calcular importancia de la columna
     importanciaC=comprobarColumna(fila,columna);
       // comprobar si hay que calcular diagonal izquierda
     if (enDiagonalIzda(fila,columna)){
          diagonalIzda=comprobarDiagonalIzda();
     }
       // comprobar si hay que calcular diagonal derecha
     if (enDiagonalDcha(fila,columna)){
          diagonalDcha=comprobarDiagonalDerecha();
     }
 // quedarse con la mejor puntuación

    //quedarse con la mejor diagonal
     if (esMejorJugada(diagonalIzda,diagonalDcha)){
          importanciaCasilla=diagonalDcha;
          } else importanciaCasilla=diagonalIzda;
    // comprobar si es mejor la fila
     if (esMejorJugada(importanciaCasilla,importanciaF)){
          importanciaCasilla=importanciaF;
          }
     // comprobar si es mejor la columna
          if (esMejorJugada(importanciaCasilla,importanciaC)){
          importanciaCasilla=importanciaC;
          }
// devolver mejor puntuación
    return importanciaCasilla;
}
```

37. Subrutina colocarFicha:

Variables de Entrada: WORD importancia.
Variables de Salida: BOOL.
Este método busca en todas las celdas vacías del tablero una celda con la importancia que se le ha pasado como parámetro. Si la encuentra inserta la ficha y finaliza. Si no encuentra ninguna celda con esa importancia devuelve FALSE.
Para ello creamos una flag booleana (BOOL insertada) que nos avisa si se ha insertado la ficha en una celda con esa importancia, recorremos el tablero con dos for y vemos en las celdas que están vacías si tienen esa importancia:

```
  if (buscaMejorOpcion(i,j)==importancia){
     insertarFicha(i,j);
     insertada=TRUE;
```

38. Subrutina copiaArray:

Variables de Entrada: BYTE leerArray[MAXFILAS][MAXCOLS], BYTE copiarArray [MAXFILAS][MAXCOLS] .
Variables de Salida: void.
Copia el array leerArray en copiarArray. Nos sirve para hacer copias del tablero, lo que hace simplemente es:

```
for (i=0;i < MAXFILAS;i++){
     for (j=0;j < MAXCOLS;j++){
          copiarArray[i][j]=leerArray[i][j];
     }
}
```

39. Subrutina jugadaAutomatica:

Variables de Entrada: void.
Variables de Salida: WORD importancia.
Retorna la importancia con la que se colocó la ficha.

- Creamos un array (imporatancias[NUMIMPORTANCIAS]) que contiene:
  importancias[0]=GANA;// el jugador gana.
  importancias[1]=EVITAPERDER;// evita que gane el otro.
  importancias[2]=SIPUEDEGANARESQUINA;// puede ganar en la próxima por esquina.
  importancias[3]=SIPUEDEGANAR;// puede ganar en la próxima.
  importancias[4]=NOPUEDEGANARESQUINA;// no puede ganar en la próxima por esquina.
  importancias[5]=NOPUEDEGANAR;// no puede ganar en la próxima.
- Hacemos una flag: *colocada* booleana que nos dice si hemos colocado la ficha.
- Intentamos colocar fichas desde la máxima importancia (GANA) a la última (NOPUEDEGANAR). Cuando logra colocar alguna ficha con la importancia determinada (*colocada=TRUE*) o cuando ha intentado ya todas las importancias se para.
- Retornamos la importancia.

40. Subrutina fichaSeMueve:

Variables de Entrada: WORD fila, WORD columna.
Variables de Salida: BOOL.
Variables globales: BYTE tablero[MAXFILAS][MAXCOLS].
Comprueba que la ficha se mueve. Sirve para evitar los "falsos EVITAPERDER", ya que si la ficha está tapando un tres en raya al quitarla y llamar a jugadaAutomatica() si no hay ninguna opción de ganar en otra celda nos devolverá EVITARPERDER como mayor importancia y se colocará la ficha en el mismo sitio, por lo tanto habría situaciones en las que la máquina no mueve.

```
quitarFicha(fila,columna);
importanciaNueva = jugadaAutomatica();
// comprobar que se ha insertado en la mismo lugar
if (tablero[fila][columna] ==simbolo(turnoActual)){
          copiarArray(tempTablero,tablero);// restaurar tablero
           return FALSE;
} else {
     copiarArray(tempTablero,tablero);// restaurar tablero
      return TRUE;
}
}
```

41. Subrutina jugadaContrario:

Variables de Entrada: void.
Variables de Salida: WORD mejoImportancia.
Variables globales: WORD turnoActual.

42. Subrutina jugadaAutomaticaDesplaza:

Variables de Entrada: void.
Variables de Salida: WORD importancia.
Variables globales: WORD turnoActual.
Jugada de la maquina cuando tiene que desplazar. Se juega probando desplazar cada ficha de la maquina, se almacena la mejor importancia y finalmente se restaura el tablero con el mejor resultado.

43. Subrutina ajustaTimer1:

Variables de Entrada: void.
Variables de Salida: void.
Inicializa la interrupción del Timer1 y arranca el timer1

44. Subrutina ajustaTimer0:

Variables de Entrada: WORD trr.
Variables de Salida: void.
Reprograma el timer para reproducir una nueva nota.

45. Subrutina calcularPeriodo:

Variables de Entrada: WORD frecuenciaNota.
Variables de Salida: WORD temp.
Variables globales:
Sirve para calcular el nuevo periodo del timer0 para generar la onda cuadrada. Retorna el valor de comparación del timer (TRR0). T=1/(2f) (frecuenciaNota = frecuencia del tono de la nota).

46. Subrutina toutMelodia:

Variables de Entrada: void.
Variables de Salida: void
Variables globales: WORD duracionNotaActual, WORD duracionTotalNotaActual.
Gestiona la melodía de fondo. Si debe cambiar de nota llama a cambiadota():

```
if (duracionNotaActual>tiempoMelodia[duracionTotalNotaActual]){
       cambiaNota();
}
```

47. Subrutina toutPerdedor:

Variables de Entrada: void.
Variables de Salida: void
Variables globales: WORD duracionNotaActual, WORD duracionTotalNotaActual.
Gestiona la melodía del perdedor. Si debe cambiar de nota o pararse llama a cambiaNotaPerdedor():

```
if (duracionNotaActual>timeMelodiaPerdedor[duracionTotalNotaActual]){
        cambiaNotaPerdedor();
}
```

48. Subrutina toutGanador:

Variables de Entrada: void.
Variables de Salida: void.
Variables globales: WORD duracionNotaActual, WORD duracionTotalNotaActual.
Gestiona la melodía del ganador. Si debe cambiar de nota o pararse llama a cambiaNotaGanador():

```
if (duracionNotaActual>timeMelodiaGanador[duracionTotalNotaActual]){
        cambiaNotaGanador();
}
```

49. Subrutinas cambiaNotaGanador, cambiaNotaPerdedor:

Variables de Entrada: void.
Variables de Salida: void.
Variables globales: WORD notaActual, WORD duracionNotaActual,
WORD duracionTotalNotaActual.
Estas rutinas cambian una nota de la melodía de ganador o de la melodía de perdedor respectivamente. Vemos el código de una de ellas porque son equivalentes:

```
void cambiaNotaGanador(void){
       notaActual++;
         duracionTotalNotaActual++;
         duracionNotaActual=0;
         if (notaActual>MAX_TONOS_GANADOR-1){
            //Parar la melodia
            suenaGanador=FALSE;
            ajustaTimer0(0);
         } else {
            ajustaTimer0(calculaPeriodo(melodiaGanador[notaActual]));
         }
}
```

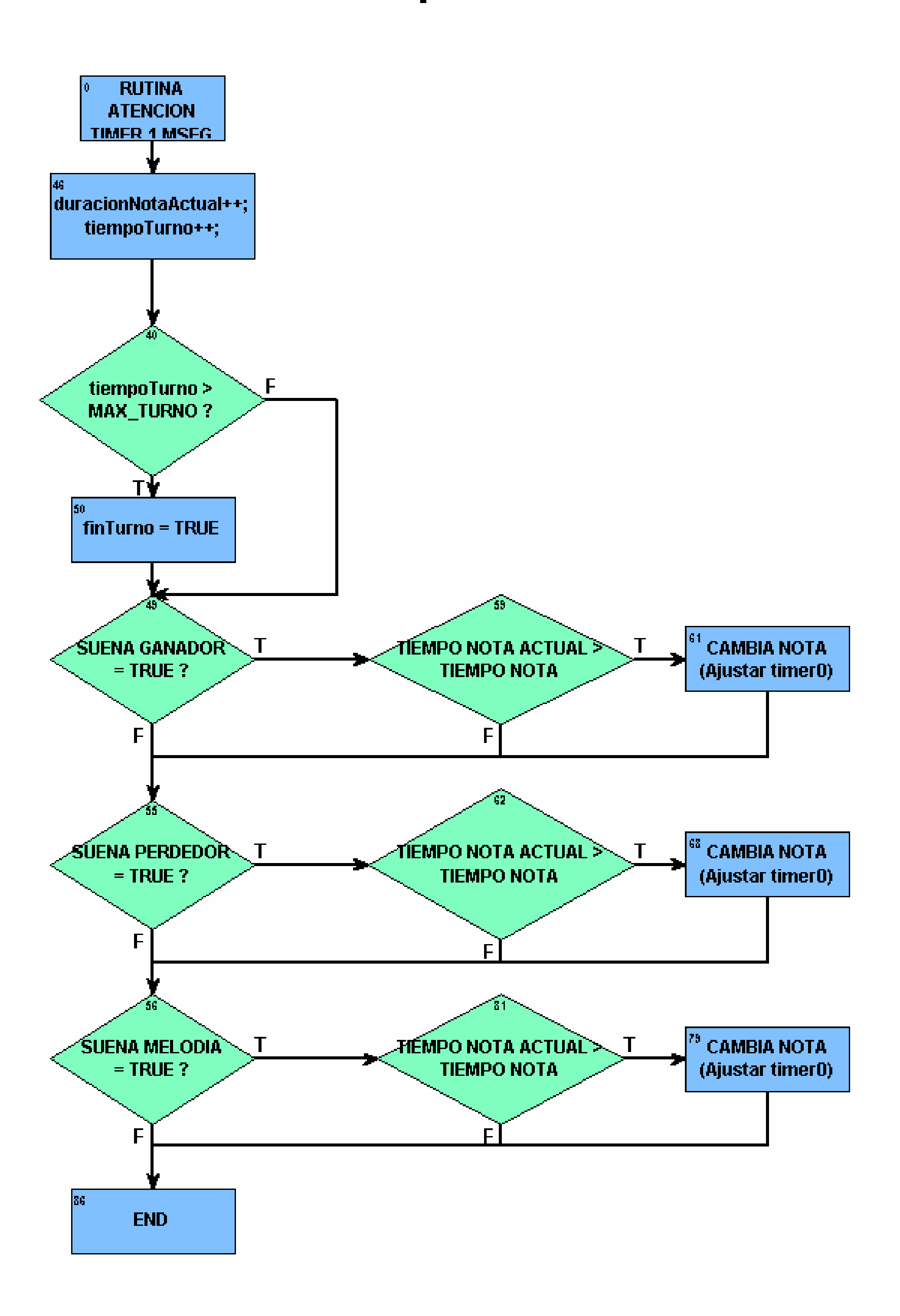
50. Subrutina cambiaNota:

Variables de Entrada: void.
Variables de Salida: void.
Variables globales: WORD notaActual, WORD duracionNotaActual,
WORD duracionTotalNotaActual.
Cambia una nota de la melodía de fondo.

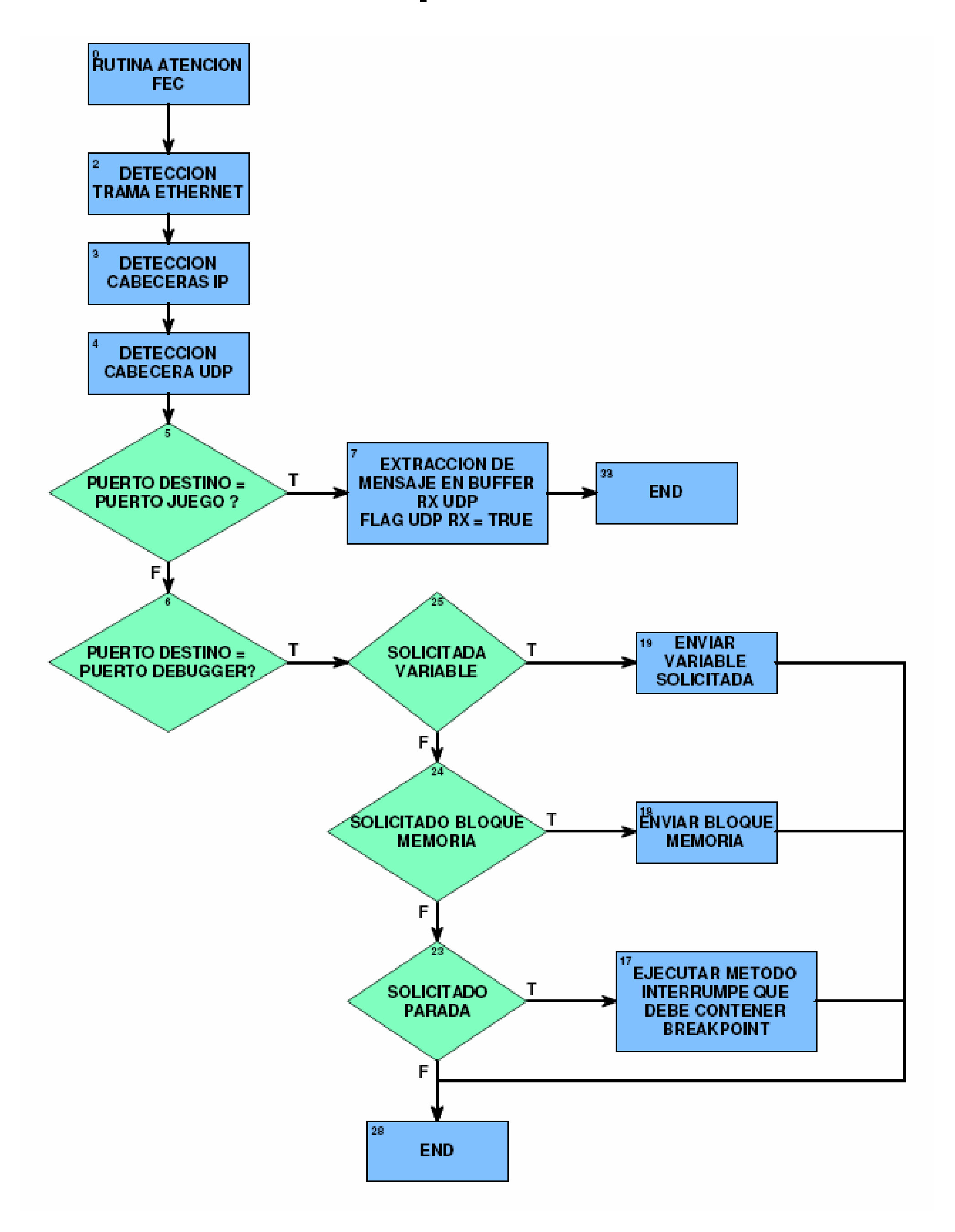
```
void cambiaNota(void){
       notaActual++;
         duracionTotalNotaActual++;
         duracionNotaActual=0;
         if (notaActual>MAX_TONOS_MELODIA-1){
            //volver al inicio de la melodia
            notaActual=0;
            duracionTotalNotaActual=0;
         }
         // reprogramar timer con la nueva tecla
          ajustaTimer0(calculaPeriodo(melodia[notaActual]));

}
```

4.2 Proceso de la interrupción timer 1.



4.3 Proceso de la interrupción FEC.



5 Descripción de las mejoras

Hemos realizado las siguientes mejoras sobre la práctica básica:

- Mejora del algoritmo de juego de la máquina.
- Desarrollo del subsistema Hardware sobre PCB
- Desarrollo de una aplicación sobre PC para dotar al sistema de entorno gráfico.
- Uso del módulo ethernet e implementación de una librería UDP.
- Implementación gráfica de estadísticas de juego.
- Desarrollo de un terminal UDP en PC para probar la funcionalidad de los sistemas (plataforma ENT y aplicación PC)
- Desarrollo de protocolo de comunicaciones para obtener variables de programa y bloques de memoria en tiempo de ejecución mediante módulo ethernet.

5.1 Mejora del algoritmo de juego de la máquina.

Tras analizar el algoritmo propuesto por la práctica decidimos intentar mejorar su capacidad de juego. En lugar de jugar de una forma tan simple como la propuesta procuramos dotar a la máquina de mayor inteligencia.

Definimos distintos niveles de importancia a cada tipo de jugada. La máquina prueba todas las posibilidades existentes de juego y posteriormente elige la jugada de mayor importancia. Una primera clasificación de las importancias definía los siguientes tipos de jugada:

IMPORTANCIA	DENOMINACION	TIPO DE JUGADA
4	GANA	Realizando esta jugada gana la partida
3	EVITA PERDER	Evita que el contrario consiga ganar en la siguiente jugada
2	SI PUEDE GANAR	Si juega en esta posición tiene posibilidades de ganar en su siguiente turno ya que tendría 2 fichas en posición ganadora y la tercera casilla asociada a esa jugada está libre.
1	NO PUEDE GANAR	Resto de jugadas. En principio la máquina no podrá ganar en el próximo turno en los segmentos correspondientes a esa casilla porque el contrario tiene insertadas fichas en esos segmentos.

Posteriormente nos dimos cuenta de que aunque el algoritmo diseñado se aplicaba correctamente, la máquina tenia todavía una estrategia muy simple y era fácilmente superable.

Entonces decidimos que en cada jugada de la máquina y antes de asignársele la importancia definitiva a esa casilla, la máquina jugara con las fichas del contrario y comprobara si este podría ganar de alguna forma. Si esto ocurría, la importancia de la jugada analizada se ponía al mínimo nivel, siempre y cuando la importancia inicial fuese 4 (ya que la máquina ganaría y daría igual lo que pueda hacer el contrario).

Continuamos mejorando el algoritmo porque nos dimos cuenta que las esquinas eran celdas con mayor importancia, puesto que tienen 2 segmentos asociados (vertical y horizontal), frente al único que tienen las celdas intermedias del perímetro exterior del tablero (celdas numero 2, 4, 6 y 8 que solo tienen horizontal o vertical). Hay que tener en cuenta que la casilla central siempre es del jugador 1 y es inamovible.

Por tanto las esquinas al tener más segmentos asociados tienen más posibilidades de obtener victoria. De esta forma definimos una nueva clasificación de las importancias asociadas a cada jugada:

IMPORTANCIA	DENOMINACION	TIPO DE JUGADA
6	GANA	Realizando esta jugada gana la partida.
5	EVITA PERDER	Evita que el contrario consiga ganar en la siguiente jugada.
4	SI PUEDE GANAR Y ES ESQUINA	Si juega en esta posición tiene posibilidades de ganar en su siguiente turno ya que tendría 2 fichas en posición ganadora y la tercera casilla asociada a esa jugada está libre. Además la celda es esquina.
3	SI PUEDE GANAR	Si juega en esta posición tiene posibilidades de ganar en su siguiente turno ya que tendría 2 fichas en posición ganadora y la tercera casilla asociada a esa jugada está libre.
2	NO PUEDE GANAR Y ES ESQUINA	Resto de jugadas. En principio la máquina no podrá ganar en el próximo turno en los segmentos correspondientes a esa casilla porque el contrario tiene insertadas fichas en esos segmentos. Además es esquina.
1	NO PUEDE GANAR	Resto de jugadas. En principio la máquina no podrá ganar en el próximo turno en los segmentos correspondientes a esa casilla porque el contrario tiene insertadas fichas en esos segmentos.

Una vez implementado este nuevo algoritmo comprobamos que en estas condiciones la máquina **ES INVENCIBLE!!.** Sin embargo en algunas ocasiones la máquina no realiza movimiento alguno. Esto es debido a que para la máquina, la jugada de mayor importancia solía ser quitar una ficha que evitaba perder y posteriormente asignarle importancia de no perder superando en importancia al resto de jugadas. Tuvimos que perfeccionar el algoritmo para que esto no ocurriera.

El resultado es un algoritmo bastante eficaz. No hemos sido capaces a día de hoy de vencer a la máquina. Analizando cada jugada que realiza esta no hemos encontrado ningún detalle que nos haga pensar que quien está jugando es una máquina, ya que no encontramos alternativa mejor a la realizada

por ella. Lo normal es que la máquina nos derrote en menos de 10 movimientos. A menos que realicemos jugadas repetitivas.

5.2 Desarrollo del subsistema Hardware sobre PCB

Conforme a lo establecido en la memoria de la práctica hemos desarrollado el hardware requerido para la implementación musical sobre un circuito PCB.

El objetivo de esta mejora es la de evitar fallos inesperados en el día de la evaluación por un posible fallo de conexionado.

Ha sido nuestra primera experiencia en este campo ya que en otros laboratorios como LECP o LCEL no habíamos optado por esta solución. La verdad es que no ha sido de gran dificultad, puesto que el software utilizado era de fácil uso.

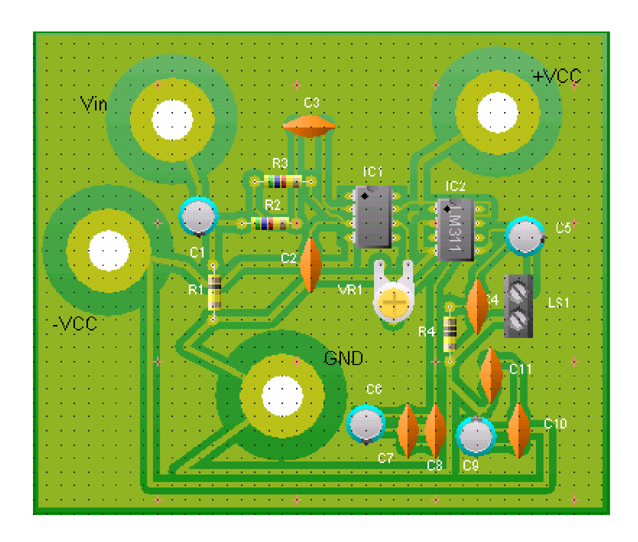
Quizás el mayor escollo haya sido el no haber encontrado las librerías de los componentes a utilizar y tener que definir nosotros mismos los dispositivos mediante librerías genéricas.

El circuito diseñado fue previamente testado utilizando Pspice. Posteriormente procedimos a realizar un prototipo sobre placas de inserción, y finalmente realizamos el montaje sobre PCB.

El montaje no presentó problema alguno excepto el hecho de utilizar un potenciómetro horizontal en lugar de los multivuelta que permiten un ajuste más fino. Nuevamente el problema para no poder utilizar este componente fue la ausencia de la librería de este tipo de componentes.

Por este motivo tuvimos que reemplazar el potenciómetro en dos ocasiones resultando un tanto complicada la extracción del mismo.

El resultado obtenido fue el esperado. Un circuito mucho más robusto que permite una manipulación segura del mismo.



5.3 Desarrollo de una aplicación sobre PC para dotar al sistema de entorno gráfico.

La principal deficiencia que presentaba nuestra práctica era lo precario del entorno. Por ello, decidimos dotar el sistema de un entorno gráfico.

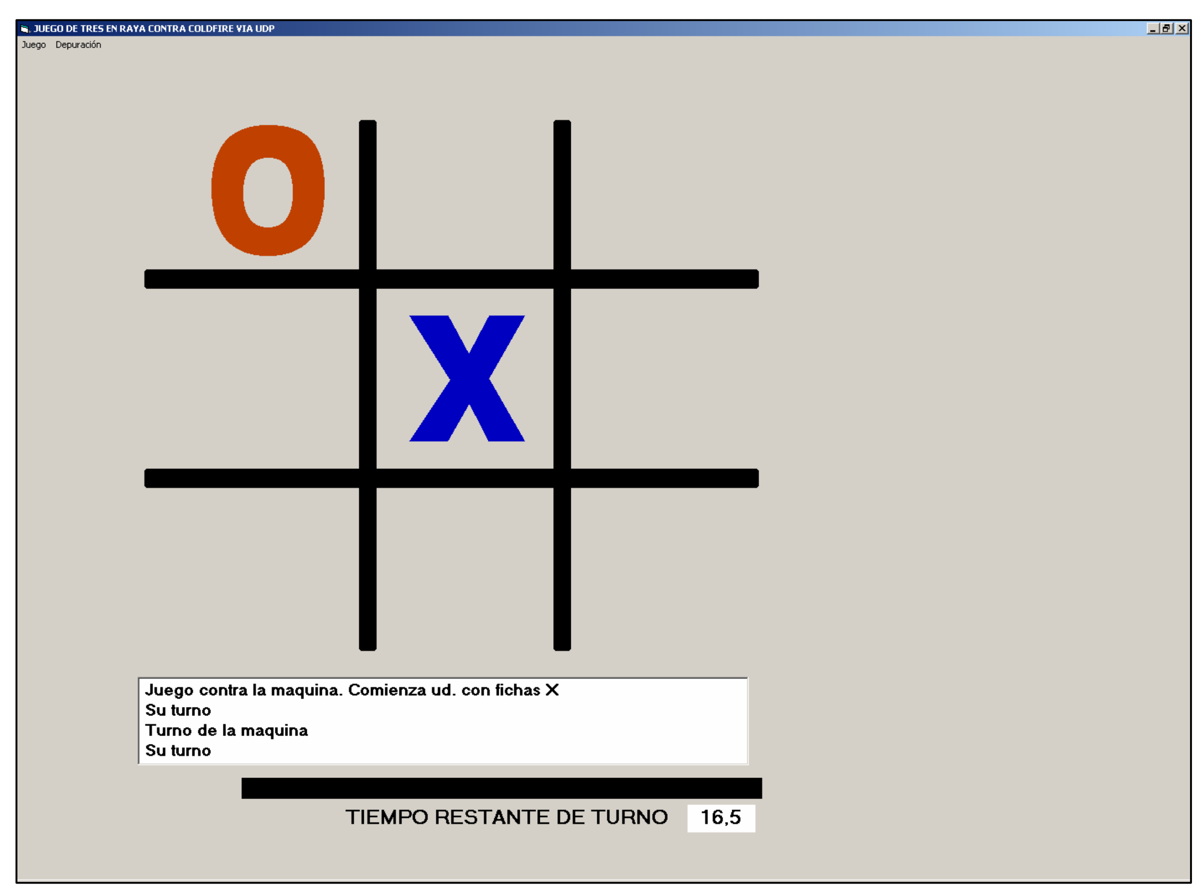
Después de analizar las posibilidades que nos ofrecía el entorno de desarrollo y las capacidades del hardware optamos por hacer uso del módulo ethernet.

Decidimos que el juego debería ser capaz de interactuar simultáneamente con la plataforma y con un terminal gráfico (PC) conectado al modulo ethernet, pudiendo cambiar indistintamente de terminal en cualquier punto de la partida.

Planteamos el intercambio necesario de datos que plataforma y terminal deberían compartir. Definimos la estructura de los mensajes y su contenido. Posteriormente desarrollamos el terminal gráfico según los requisitos obtenidos.

Optamos por desarrollar el software del terminal gráfico sobre Visual Basic 6.0 ya que presentaba mayor facilidad que Java para aplicaciones gráficas y permitía una gestión de los eventos de teclado y ratón suficientes para conseguir el objetivo propuesto.

El resultado obtenido ha cumplido con creces los objetivos fijados. El usuario dispone de una interfaz gráfica de alta calidad que interactúa de forma natural (sin retardos) con la plataforma ENT.A continuación mostramos unas capturas de pantalla del interfaz gráfico:



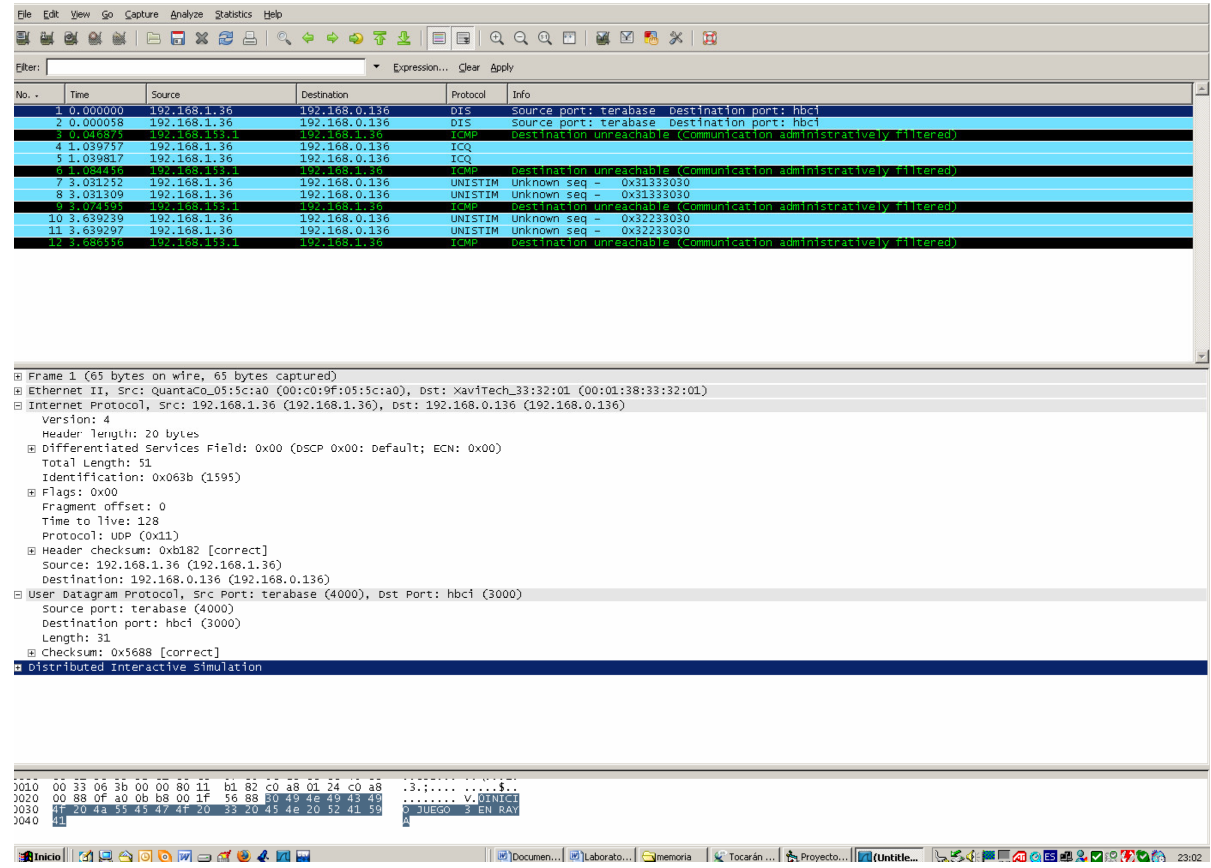
5.4 Uso del módulo ethernet e implementación de una librería UDP.

La única documentación disponible para el uso del módulo ethernet era una aplicación TFTP. Encontramos también librerías para el Coldfire de todo tipo de protocolos UDP, TCP, SNMP, DHCP etc… pero el software era licenciado y por tanto lo descartamos.

El servidor TFTP no nos parecía la mejor opción para nuestro desarrollo. No era ni mucho menos la solución más natural, pero si en cambio era la de más fácil implementación. Nos obligaba a tener que instalar un servidor FTP (Solaris). Sin embargo, por lo que fue descartado fue porque queríamos que el terminal pudiese funcionar como si de una aplicación local se tratase; sin demoras entre la intervención del usuario y la respuesta del coldfire. Un usuario puede llegar a generar múltiples eventos en un lapso de tiempo muy corto. Antes de que se produzca un nuevo evento se ha de haber recibido respuesta del anterior.

Finalmente optamos por desarrollar nuestra propia librería UDP. Éramos programadores novatos en C, y la aplicación TFTP utilizaba complejas estructuras de punteros y datos. Fue una tarea muy complicada llegar a entender el funcionamiento de la aplicación en la que invertimos mucho más tiempo del esperado pero que nos permitió aprender mucho sobre el uso de punteros.

Logramos escribir una librería que permitía enviar mensajes UDP. Para depurar esta librería hicimos uso del analizador de red WireShark para poder visualizar los mensajes enviados mostrando la cabecera UDP con todos sus campos como se muestra a continuación:



La librería permite enviar y recibir mensajes UDP por los puertos origen y destino deseados, responde a peticiones PING y permite el uso de red, mascara de red y puerta de enlace; por tanto, la plataforma podría colgarse de cualquier red existente permitiendo a un usuario remoto con conectividad IP interactuar con el coldfire. Sin embargo, para simplificar las conexiones en el laboratorio la comunicación entre el PC y la plataforma se realiza mediante un cable RJ45 cruzado.

Por último queremos destacar que la librería UDP se ha realizado de tal forma que pueda ser utilizada para cualquier otro fin distinto al juego 3 en raya. Facilita el envío y mensajes UDP de forma sencilla e intuitiva. Dispone de métodos de inicialización, envío y atención de rutinas de recepción. De esta forma puede ponerse a disposición de terceras personas para que puedan utilizar el puerto ethernet de una forma más eficaz que la ofrecida por el cliente TFTP para futuras aplicaciones.

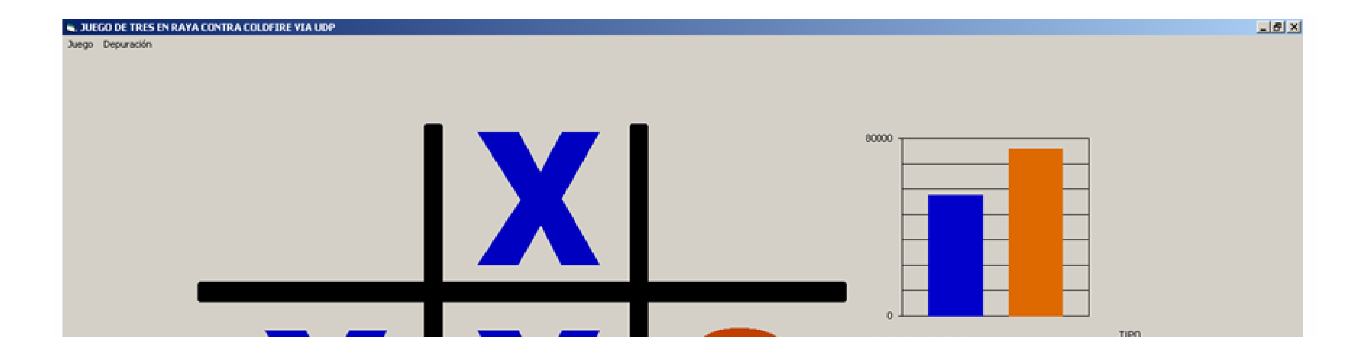
5.5 Implementación gráfica de estadísticas de juego.

Para darle más funcionalidad al entorno gráfico, hemos desarrollado un sistema que permite visualizar gráficamente el tiempo de juego acumulado de cada uno de los jugadores. Puesto que el tiempo total de una partida no es predecible y puede oscilar mucho, hemos diseñado un sistema de ejes variables en tiempo de ejecución. El resultado obtenido es una gráfica que se muestra con una escala apropiada pero tampoco varia constantemente, lo que dificultaría su lectura.

El tiempo restante de cada turno se representa además de numéricamente mediante una barra horizontal que se va reduciendo. En los últimos 5 segundos de turno el sistema avisa al jugador cambiando de color tanto la barra como el tiempo restante de negro a rojo.

La actualización de las estadísticas y del tiempo restante de turno se realiza cada 100 msegs. El software dispone de un menú de configuración que permite mostrar u ocultar las estadísticas según deseos del jugador. No se han implementado más estadísticas porque hemos considerado que era un trabajo repetitivo y tampoco aportaba mucho al desarrollo de un juego tan trivial como el tres en raya.

A continuación mostramos la captura de un trozo de pantalla de una partida con estadísticas:



5.6 Desarrollo de un terminal UDP en PC para probar la funcionalidad de los sistemas (plataforma ENT y aplicación PC)

Para evitar tener que depurar dos sistemas a la vez (entrenadora y PC) decidimos que lo mejor sería desarrollar un terminal de comunicaciones UDP que enviara mensajes formateados según el protocolo de comunicaciones diseñado hacia los dos extremos y comprobar que eran interpretados correctamente. De esta forma pudimos depurar fuera del laboratorio la aplicación Visual Basic. El terminal envía textos, tiempo restante, posiciones del tablero etc… y muestra los mensajes que genera la aplicación cuando se producen determinados eventos: el usuario pulsa una celda, arrastra una ficha o pulsa una tecla.

Posteriormente hicimos lo mismo con el coldfire. Podíamos enviarle al micro peticiones y comprobar que este interpretaba los mensajes e interactuaba con el juego.

La depuración final de todo el sistema si hizo con la ayuda de este terminal, los entornos de depuración VB y ENT, y el analizador de red WireShark.

A continuación mostramos una captura de pantalla del terminal UDP desarrollado.

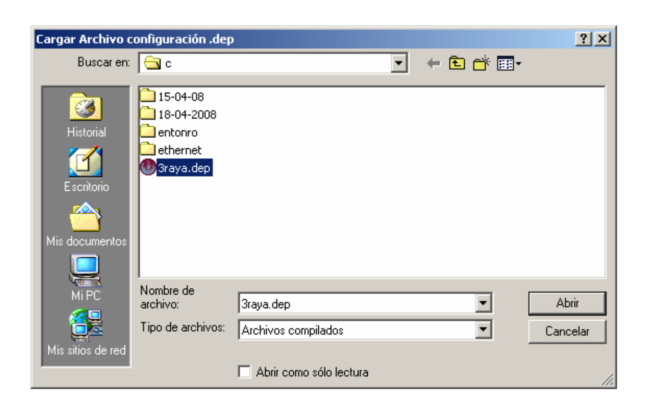
5.7 Desarrollo de protocolo de comunicaciones para obtener variables de programa y bloques de memoria en tiempo de ejecución mediante módulo ethernet.

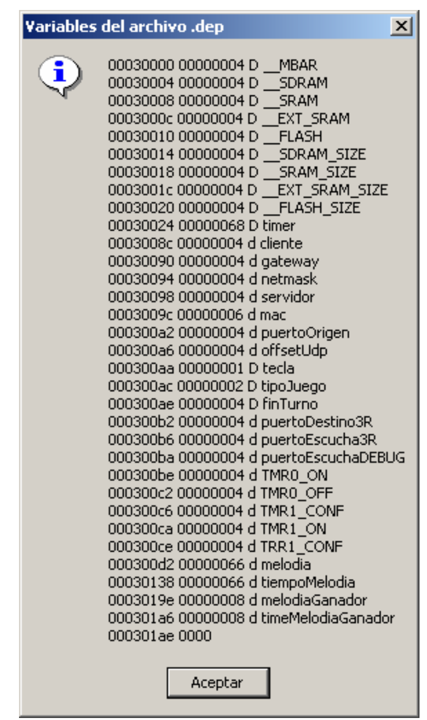
Por último, tras barajar nuevas mejoras, se nos ocurrió que ya que disponíamos de un canal de comunicación de alta velocidad (ethernet) y un entorno gráfico de alto nivel podíamos realizar algunas tareas de debug desde nuestro PC. El micro escucharía por un puerto distinto al que realiza la interacción con el juego. Los mensajes formateados son interpretados, se atiende la petición y se responde con la respuesta.

Inicialmente pensamos que podíamos hacernos con el control de la plataforma. Podríamos detenerla mediante un STOP y realizar el debug sobre la propia entrenadora, pero pronto entendimos que la entrenadora ejecuta un software rundebug que no atiende peticiones del micro de este tipo. El micro se pararía pero la entrenadora no se daría cuenta. Por tanto estábamos únicamente limitados a poder visualizar registros y variables.

Debido a que las variables locales ocupan memoria en tiempo de ejecución (por como se comporta C) solamente podemos acceder a las variables globales que en tiempo de compilación tienen asignada una posición determinada de la memoria. Otra opción del sistema es la de pedir un volcado de un bloque de memoria.

Para realizar un entorno amigable, decidimos ejecutar un "parse" sobre el archivo .DEP capturando la información relativa a la posición de las variables. Cada nombre de variable global definida en nuestro código se introduce en un menú desplegable tipo "combo". Cada etiqueta tiene asociada una dirección de memoria. Como en el archivo .DEP no figura el tamaño de la variable, el propio usuario tiene que especificar si se trata de una variable tipo LONG, WORD o BYTE.





El programa envía una petición UDP al puerto de escucha para debugging con la dirección de memoria inicial y el número de bytes a leer. El micro responde con la información solicitada.

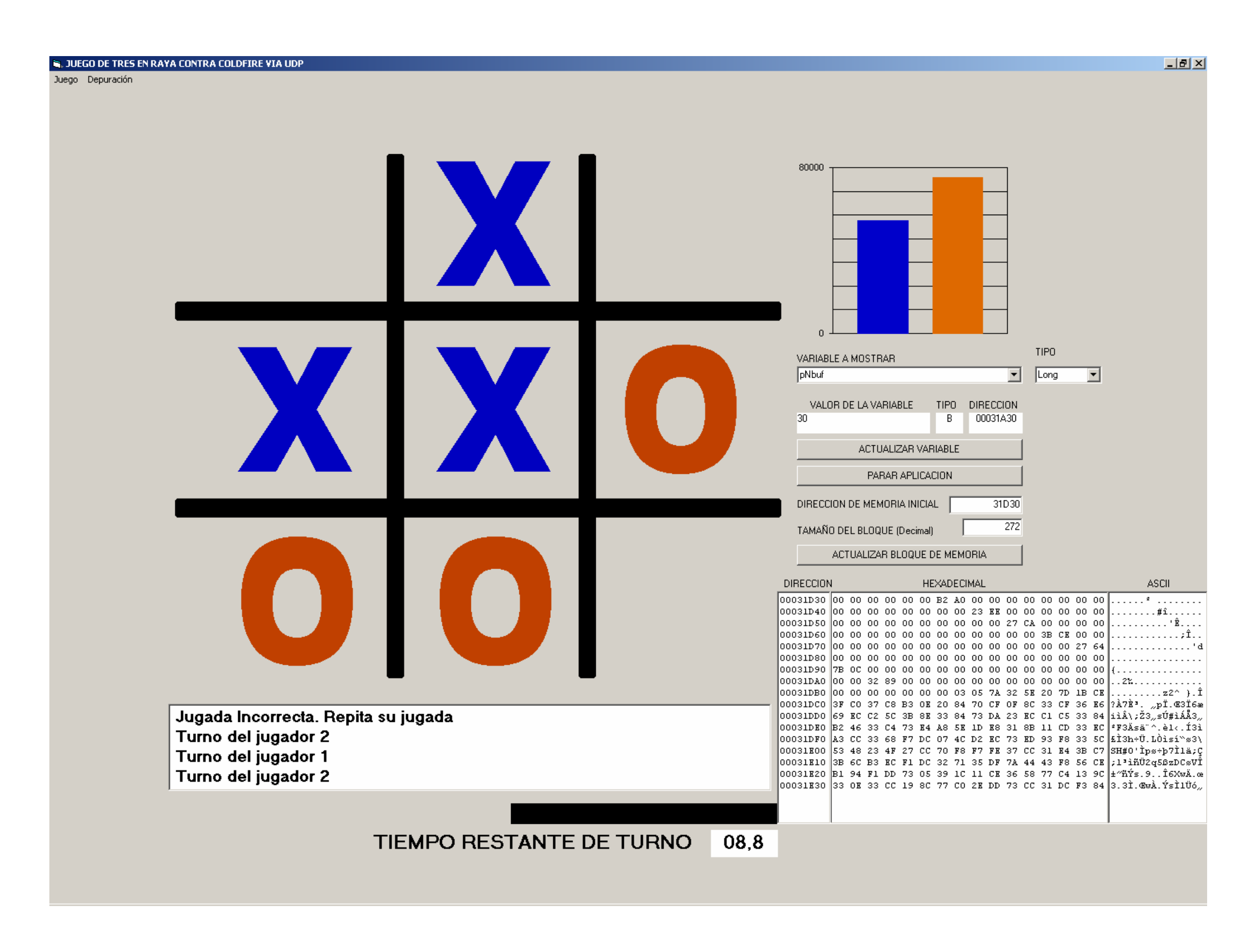
Aunque inicialmente parecía una solución sencilla de implementar, tuvimos un arduo trabajo por delante para poder manejar correctamente la información. El Posicionamiento de los bytes, la conversión

de tipos, la conversión de hexadecimal a decimal e inversa, manejo de caracteres ASCII que unas veces son caracteres y otras valores numéricos o hexadecimales.

Hacer que la información fuese manejable entre las dos plataformas no fue nada sencillo. Pero el trabajo tuvo su recompensa y el resultado es un sistema que lee bloques de memoria y variables globales de forma instantánea y en tiempo de ejecución.

Por último queremos destacar que se nos ocurrió como detener la plataforma a nuestra voluntad. Modificando un flag en el programa mediante mensaje UDP podemos hacer que una sentencia condicional (por ejemplo en la rutina de atención del timer de 1 mseg) salte a una línea donde previamente se le ha establecido un punto de parada. Todo esto quedó implementado y ha servido de gran ayuda para terminar de depurar el código.

A continuación mostramos una captura de pantalla del entorno de depuración desarrollado:



6 Principales problemas encontrados

Inicialmente nos hemos enfrentado a los típicos problemas normales en la iniciación de un nuevo entorno y sobre todo en el uso de C. La práctica básica no nos ha supuesto problemas serios. A medida que hemos ido implementando mejoras nos hemos enfrentado a problemas de mayor índole. Aprender a manejar las estructuras con punteros que utilizaban las librerías de ethernet no fue tarea fácil.

También en la mejora que incluye un entorno de debugging nos hemos enfrentado con problemas de conversión de tipos (manejo de caracteres ASCII y su conversión para direccionamientos de memoria, conversión hexadecimal a decimal etc…). En cuanto a problemas puntuales queremos destacar los siguientes:

Después de estar probando una librería ethernet y ver que no funcionaba vimos que el problema radicaba en el cable RJ45 interno de la plataforma. Estaba suelto el terminal que conecta en el circuito que integra el microcontrolador. Se nos proporcionó otra plataforma y se solucionó el problema. Tardamos más tiempo en solucionar el cambio de la plataforma que en detectar el problema.

Otro problema que fue mucho más difícil de detectar y nos hizo perder mucho tiempo (casi una sesión de laboratorio) fue detectar un fallo en el entorno de depuración. El problema se nos daba al realizar una multiplicación de dos long. El resultado era 0 cuando ambos eran no nulos. Fuimos simplificando el código para poder aislar el problema, declarando todas las variables para poder realizar un seguimiento. Visto que desde C no obteníamos respuesta bajamos a nivel ensamblador, analizando los registros implicados y el código asociado. Tampoco obtuvimos respuesta. Revisamos la documentación de la instrucción mul por si pudiera estar escapándose algo. Finalmente pensamos que la ALU podía presentar algún error. Cambiamos de plataforma y ante todo pronóstico funcionó perfectamente. Un análisis mucho más detallado con la ayuda de un profesor nos hizo dar con la solución del problema. Hicimos una ejecución simultánea en las dos plataformas, revisando el contenido de registros, CCR pila etc… Finalmente comprobamos que nuestra plataforma direccionaba 4 bytes antes la misma variable global en el WATCH. La respuesta del profesor fue que se debía a un bug del entorno. Dependiendo en que momento se hacia el WATCH la plataforma apuntaba a una dirección o a otra.

Conforme a lo estipulado en el formulario de la memoria procedemos a realizar la siguiente baremación:

Módulos	Puntuación máxima	Puntuación esperada
Requisitos mínimos	**35**	35
Memoria	**15**	10
Software	**25**	20
Hardware	**5**	4
Mejoras	**20**	20
TOTAL	**100**	89

En cuanto a la nota final del proyecto creemos que hemos realizado un buen trabajo cumpliendo con creces los requisitos de la práctica propuesta. Hemos sido capaces de manejar con soltura la plataforma, defendernos bastante bien en el uso de C y en el manejo de complicadas librerías de C.

A nuestro juicio, creemos que hemos propuesto unas mejoras creativas, que no buscaban ser repetitivas ni monótonas. Más allá de todo esto consideramos que debe ser el profesorado quien establezca la nota final de la práctica.

7 Manual de usuario

El sistema desarrollado permite la interacción del usuario tanto desde la plataforma ENT como desde un terminal PC.

Toda operación disponible desde la plataforma ENT está también disponible desde el entorno PC. En cualquier momento de la ejecución del programa se puede conmutar entre un sistema y el otro, pudiendo por ejemplo disputar una partida a 2 jugadores. Uno desde el Pc y otro desde la plataforma.

El PC con el software del 3 en raya instalado, deberá tener conectividad IP con la plataforma. La forma más sencilla de implementar dicha conectividad es mediante un cable de red cruzado (pines 1 a 3 y 2 a 6 del conector RJ45). El PC deberá tener la siguiente dirección IP en su interfaz de red: 192.168.1.36 /24.

Inicialmente el sistema muestra un mensaje de bienvenida mostrando un menú que permite jugar contra la máquina (pulsando la tecla A del teclado matricial o del PC) o en modo dos jugadores (pulsando la tecla B).

El sistema va informando por pantalla al usuario de la tarea que tiene que realizar y muestra los mensajes de error asociados a las maniobras incorrectas realizadas por este.

Para poder referirse a una celda o casilla del tablero desde el teclado matricial existe la siguiente relación que muestra la tecla asociada a cada celda del tablero dibujada.

1	2	3
4	5	6
7	8	9

De esta forma, para seleccionar la celda central se deberá pulsar 5. Desde el PC se pulsará sobre la celda con el ratón o se arrastrará la ficha que se desee a su destino final.

Una vez finalizada la partida el sistema pregunta si se desea volver a jugar o finalizar la aplicación.

Desde el PC existen menús adicionales. Uno de ellos muestra información estadística del tiempo de juego y otra despliega un menú de depuración que permite mostrar variables globales y bloques de memoria. Su uso es muy intuitivo. Una vez que seleccionamos mostrar depuración el programa nos pregunta por la ubicación del archivo .DEP que deberá ser el mismo que el cargado en la plataforma y contiene las direcciones de memoria de las variables.

8 Bibliografía

1. R. San Segundo, "Introducción a los Sistemas Digitales con el microcontrolador MCF5272" Ed. Marcombo S.A. ETSIT, 2006.

• Utilizado para llevar a cabo la práctica del MCF5272, dado que aparecen varios tutoriales para el manejo de los principales recursos de la plataforma ENT2004CF, tales como temporizadores, teclado…

2. R. San Segundo, "Entorno de desarrollo EDColdFire V3.0" Ed. ETSIT, 2006b.

• Nos ha sido de utilidad sobretodo al inicio del desarrollo del proyecto, dado que describe el manejo del programa EDColdFire y contiene ejemplos sencillos para el manejo de la plataforma ENT2004CF, por ello nos ayudo a conocer el manejo tanto de la plataforma como del programa más rápidamente.

3. Javier Ferreiros, "Aspectos Prácticos de Diseño y Medida en Laboratorios de Electrónica" Ed. ETSIT. 2002

4. Wikipedia.org. Diseño de filtro RC

5. J.M Montero et al. "Dudas y preguntas frecuentes del LSED" Disponible en Internet a través del portal del LSED http://lorien.die.upm.es/lsed/

6. C. Carreras "Sistemas Electrónicos Digitales" Ed. ETSIT. 2005

• Libro utilizado en la asignatura de SEDG, utilizado para dudas en cuanto a programación en ensamblador, ya que contiene información acerca de las diferentes aplicaciones y recursos que se pueden obtener del MCF5272, puesto que en él se tratan todos los temas básicos que caracterizan a dicho controlador.

7. Hoja de características del LM386 y LM356.

8. Memoria LCEL año 2007/2008.

9. Dr Sydney Feit, "TCP/IP Arquitectura, protocolos e implementación con IPv6 y seguridad IP" Ed. McGraw-Hill.

• Ha sido nuestro libro de referencia para para diseñar las comunicaciones ethernet y consultar la estructura de los datagramas IP, UDP y tramas Ethernet.

9 ANEXO I: Código del programa de la primera sesión

```
#include "m5272lib.c"
#include "m5272gpio.c"
#define NUM_FILAS 4 //teclado matricial
#define NUM_COLS 4 //teclado matricial
#define EXCIT 1
#define RETARDOTECLA 1150
#define UNJUGADOR 1
#define DOSJUGADORES 2
#define NULO 0 //valor de retorno nulo
#define MAXFILAS 3//filas tablero
#define MAXCOLS 3 //columnas tablero
#define MAXFICHAS 9 //max numero fichas en tablero
#define VACIO '_' //celda tablero vacia
#define FALSE 0
#define TRUE 1
#define FICHAS1 'X' // fichas con las que juega el jugador uno
#define FICHAS2 'O' // fichas con las que juega la maquina o player2
#define BARRA '\\'
// declaración de variables
BYTE tecla='A'; // tecla capturada del teclado
WORD tipoJuego=0; //(=1 con la máquina . =2 con otro jugador)
BYTE tablero[MAXFILAS][MAXCOLS]; //tablero de partida
WORD numeroFichas; //numero fichas en el tablero
WORD numFila,numColumna; //fila y columna seleccionada
WORD turnoActual; // nº de jugador que tiene su turno
```

```
//-------------------------------------------------------
// char teclado(void)
//
// Descripción:
//   Explora el teclado matricial y devuelve la tecla
//   pulsada
//-------------------------------------------------------
BYTE teclado(void)
{
  BYTE tecla;
  BYTE fila, columna, fila_mask;
  static char teclas[NUM_FILAS][NUM_COLS] = {{"123C"},
                             {"456D"},
                             {"789E"},
                             {"A0BF"}};
  // Bucle de exploración del teclado
  while(TRUE){

    // Excitamos una columna
    for(columna = NUM_COLS - 1; columna >= 0; columna--){
      set_puertoS(EXCIT << columna);          // Se envía la
excitación de columna
      retardo(RETARDOTECLA);                  // Esperamos respuesta
de optoacopladores

      // Exploramos las filas en busca de respuesta
      for(fila = NUM_COLS - 1; fila >= NULO; fila--){
        fila_mask = EXCIT << fila;       // M?scara para leer el bit
de la fila actual
        if(lee_puertoE() & fila_mask){        // Si encuentra tecla
pulsada,
          while(lee_puertoE() & fila_mask);   //   Esperamos a que se
suelte
          retardo(RETARDOTECLA);              //   Retardo
antirrebotes
          tecla=teclas[fila][columna];
          outch(tecla); //eco caracter
          output("\r\n");
       return tecla;//   Devolvemos la tecla pulsada
        }
      }
      // Siguiente columna
    }
    // Exploración finalizada sin encontrar una tecla pulsada
  }
  // Reiniciamos exploración
}
```

```
//-----------------------------------------------------------
//Metodo que selecciona tipo de Juego.
//devuelve 1= contra la maquina y 2 = 2 jugadores y 0 si no es
seleccion correcta
//-----------------------------------------------------------

void selTipoJuego(BYTE tecla){
    tipoJuego=NULO;
    if (tecla=='A'){
         tipoJuego=UNJUGADOR;
          output("\r\n Juego contra la maquina. Comienza ud. con
fichas X \r\n");
    }
    if (tecla=='B'){
         tipoJuego=DOSJUGADORES;
          output("\r\n Juego entre dos jugadores. Comienza el jugador
X \r\n");
    }
return;
}
```

```
//---------------------------------------------------------
//  void mostrarMenu(void)
//  muestra el menu del juego
//---------------------------------------------------------
void mostrarMenu(void){
 output("Elija el tipo de juego desdeado: (A) contra la máquina (B)
entre dos Jugadores:");

// selecionar tipo juego
do{
   selTipoJuego(teclado());
   if (tipoJuego==0) {
      output("\r\n Selección incorrecta. Repita selección \r\n");
   }

}while(tipoJuego==0);

return;
}
```

```
//-----------------------------------------------------------
//  metodo inicial de la platafora ENT
//-----------------------------------------------------------
void bucleMain(void){
    BOOL fin=FALSE;
    BOOL eleccion=FALSE; //comprueba que se eligio o A o B
    do{
     // mensaje inicial
     output("**************      Juego de Tres en Raya
***************\r\n");
    mostrarMenu();
     iniciaTablero();
     muestraTablero();
    jugarPartida();
       // otra partida
         output("\r\n************* ERES UN PASTOR.....¿OTRA PARTIDA?
A=SI  B=NO ***********\r\n");
     fin=FALSE;
        eleccion=FALSE;
     do{
         if (teclado()=='B'){
            fin=TRUE ;
            eleccion=TRUE;
         } else if(teclado()!='A'){
            output ("\r\nEleccion Incorrecta\r\n");
            eleccion=TRUE;
         }
     }while(eleccion==FALSE);

    }while(fin==FALSE);

    exit(0);


}
 // fin bucleMain
```

```
//-----------------------------------------------------------
// void __init(void)
//
// Descripción:
//   Funci?n por defecto de inicialización del sistema
//
//-----------------------------------------------------------
void __init(void)
{

}
```

```
//-----------------------------------------------------------
// Definici?n de rutinas de atenci?n a la interrupción
// Es necesario definirlas aunque est?n vac?as
void rutina_int1(void){}
void rutina_int2(void){}
void rutina_int3(void){}
void rutina_int4(void){}
void rutina_tout0(void){}
void rutina_tout1(void){}
void rutina_tout2(void){}
void rutina_tout3(void){}
```

10 ANEXO II: Código del programa del proyecto final

10.1 CODIGO VISUAL BASIC APLICACIÓN 3 EN RAYA

```
Dim tmpX As Integer
Dim tmpY As Integer
Dim tOut As Integer
Dim desplazandoFicha As Boolean
Dim turnoActual As Integer
Dim tiempoJugador1 As Long
Dim tiempoJugador2 As Long
Sub leeFicheroConf()

       'Fijar las propiedades necesarias
    With CommonDialog1
        .DialogTitle = "Cargar Archivo configuración .dep"
        .DefaultExt = "DEP"
        .Filter = "Archivos compilados|*.DEP"
        .Flags = cdloFNFileMustExist
        .ShowOpen 'para abrir un archivo existente

             'abrir fichero para lectura por el canal 1
    End With
capturando = False
' Get a free file number
nFileNum = FreeFile

' Read the contents of the file
Do While Not EOF(nFileNum)
   Line Input #nFileNum, sNextLine
   'do something with it
   'add line numbers to it, in this case!
   If InStr(1, sNextLine, "TABLA DE SIMBOLOS") Then
        capturando = True
        ' quitar las 2 siguientes lineas para ponerse en la primera
posicion
        Line Input #nFileNum, sNextLine
        Line Input #nFileNum, sNextLine
   End If
   If capturando = True Then
      sNextLine = sNextLine & vbCrLf
      STEXT = STEXT & sNextLine
      analizalinea (sNextLine)
   End If

Loop
```

```
MsgBox STEXT, vbInformation, "Variables del archivo .dep"


' CLng("&H" & "30A")
' Close the file
Close nFileNum

Me.variable.ListIndex = 0


End Sub
Sub analizalinea(linea As String)

finCampo1 = InStr(1, linea, " ")
fincampo2 = InStr(finCampo1 + 1, linea, " ")
fincampo3 = InStr(fincampo2 + 1, linea, " ")
fincampo4 = Len(linea)
campo1 = Mid$(linea, 1, finCampo1 - 1)
campo2 = Mid$(linea, finCampo1 + 1, fincampo2 - finCampo1 - 1)
If fincampo3 > 0 Then
    campo3 = Mid$(linea, fincampo2 + 1, fincampo3 - fincampo2 - 1)
Else
    fincampo3 = fincampo2
End If
If (fincampo4 - fincampo3) > 0 Then
    campo4 = Mid$(linea, fincampo3 + 1, fincampo4 - fincampo3 - 2)
Else
    campo4 = ""
End If
Me.variable.AddItem campo4
Me.Combo2.AddItem campo3
Me.Combo3.AddItem campo2
Me.Combo4.AddItem campo1
End Sub
Sub mensajeDebugger(dato As String)
Dim direccionvariable As Long
Dim s As String
' detectar tipo mensaje

Select Case Val(Mid$(dato, 1, 1))
    Case 1
        ' INFORMACION DE UNA VARIABLE
         ' formato= 1#valor
          Me.valorvariable.Caption = ""
          For i = 1 To Len(dato) - 2
              s = Hex(Asc(Mid$(dato, 2 + i, 1)))
              Select Case Val(s)
                   Case 0
                    ' ES UNA LETRA O UN 0
                     If s = "0" Then
```

```
                         Me.valorvariable.Caption =
Me.valorvariable.Caption + "00"
                     Else
                        Me.valorvariable.Caption =
Me.valorvariable.Caption + s
                     End If
                   Case Else
                        Me.valorvariable.Caption =
Me.valorvariable.Caption + Format(s, "0#")
              End Select
          Next
      Case 2
        ' bloque de memoria
        ' formato= 2#datos
          'Me.dir.Text = Mid$(dato, 3, 8)
          lmensaje = Len(dato) - 2

          direccionvariable = CLng("&H" & Me.memini.Text)
          'CLng("&H" & Mid$(dato, 3, 8))
          Me.dir.Text = ""
          Me.ascii.Text = ""
          Me.hexa.Text = ""
          For a = 0 To (Int((lmensaje - 1) / 16))
             ' rellenar direccion bloque memoria
             cadena2 = Hex(direccionvariable + (a * 16))
             cadena = "00000000"
             Mid$(cadena, 9 - Len(cadena2), Len(cadena2)) = cadena2
             Me.dir.Text = Me.dir.Text & cadena & vbCrLf

             ' rellenar datos en hexadecimal
             For i = 0 To 15
                 If (a * 16 + i) < lmensaje Then
                     cadena = "00"
                     cadena2 = Hex(Asc(Mid$(dato, 3 + a * 16 + i,
1)))
                     Mid(cadena, 3 - Len(cadena2), Len(cadena2)) =
cadena2
                    Me.hexa.Text = Me.hexa.Text & cadena & " "
                End If
             Next
                   Me.hexa.Text = Me.hexa.Text & vbCrLf
             ' rellenar datos en ascii
             TEXTO = ""
             For i = 0 To 15
                 If Asc(Mid$(dato, 3 + a * 16 + i, 1)) > 31 Then '
mprimible

                       TEXTO = TEXTO + Mid$(dato, 3 + a * 16 + i, 1)


                 Else
```

```
                      TEXTO = TEXTO & "."
                 End If
             Next
             Me.ascii.Text = Me.ascii.Text & TEXTO & vbCrLf

          Next ' fin linea total
 End Select
End Sub

Sub mostrarTablero(dato As String)
For a = 1 To 9
  X(a).Visible = False
  celda(a).Visible = True
Next

For a = 2 To Len(dato)
      Call Me.ponerFicha(Mid$(dato, a, 1), a - 1)
Next
End Sub


Sub ponerFicha(ficha As String, ind As Integer)
If ind > 9 Then Exit Sub
' calcular coordenadas
  Select Case ind
        Case 1, 4, 7
          X(ind).Left = 3360
        Case 2, 5, 8
          X(ind).Left = 6960
        Case 3, 6, 9
          X(ind).Left = 10320
    End Select

    Select Case ind
        Case 1, 2, 3
          X(ind).Top = 480
        Case 4, 5, 6
          X(ind).Top = 3840
        Case 7, 8, 9
          X(ind).Top = 7320
    End Select
' mostrar ficha o celda vacia
Select Case Asc(ficha)
  Case 88, 120:
   ' ficha X
   X(ind).ForeColor = &HC00000    ' azul
   X(ind).Caption = "X"
   X(ind).Visible = True
   celda(ind).Visible = False
  Case 79, 48, 111:
  ' ficha O
```

```
   X(ind).ForeColor = &H40C0&     ' naranja
   X(ind).Caption = "O"
   X(ind).Visible = True
   celda(ind).Visible = False
  Case Else
    X(ind).Visible = False
    celda(ind).Visible = True

End Select







End Sub
Sub mensaje(dato As String)
' detectar tipo mensaje

Select Case Val(Mid$(dato, 1, 1))
    Case 0
       'texto para mostrar en pantalla
       Call mostrarTexto(dato)
       ' sirve para escapar de evento mousemove de la ficha por si ha
habido error
       If desplazandoFicha = True Then
           desplazandoFicha = False
       End If
    Case 1
       ' posicion del tablero
       If desplazandoFicha = False Then
          Call mostrarTablero(dato)
        End If

    Case 2
       ' tiempo restante del turno
       Call ajustaTimer(dato)

    Case 3
       ' turno actual
        Select Case Val(Mid$(dato, 2, 1))
            Case 0 ' inicializar contadores de turnos
                grafica.Column = 1
                grafica.Data = 0
                grafica.Column = 2
                grafica.Data = 0
                tiempoJugador1 = 0
                tiempoJugador2 = 0

            Case 1
               turnoActual = 1
```

```
            Case Else
                turnoActual = 2
        End Select

End Select

End Sub
Sub mostrarTexto(dato As String)
Me.LINEAMENSAJE.Text = Me.LINEAMENSAJE.Text + Chr$(13) + Chr$(10) + "
" + Mid$(dato, 2, Len(dato) - 1)
Me.LINEAMENSAJE.SelStart = Len(Me.LINEAMENSAJE.Text)
End Sub

Private Sub bloquerefresh_Click()

' formato envio peticiones  "2#dirmem#long" dir=8 bytes long = 8
bytes
Dim dir As String

' obligar a pedir bloques de 16 en 16
tamaño.Text = Int(tamaño.Text / 16) * 16

dir = "00000000"
Mid$(dir, 9 - Len(Me.memini.Text), Len(Me.memini.Text)) =
Me.memini.Text
TEXTO = "2#" & dir & "#"
cantidad = Hex(Val(Me.tamaño.Text))
dir = "00000000"
Mid$(dir, 9 - Len(cantidad), Len(cantidad)) = cantidad
TEXTO = TEXTO & dir

Me.UDPDEBUGGER.RemotePort = 5000
Me.UDPDEBUGGER.RemoteHost = "192.168.1.20"

Me.UDPDEBUGGER.SendData (TEXTO)

End Sub

Private Sub celda_Click(Index As Integer)
Dim tecla As String

tecla = Chr$(Index + Asc("0"))
Me.UDP.SendData ("#" + tecla)

End Sub

Private Sub Combo1_Change()

End Sub

Private Sub Command1_KeyPress(KeyAscii As Integer)
```

```
Me.UDP.SendData "#" + Chr$(KeyAscii)
End Sub


Private Sub Command2_Click()
' formato envio peticiones  "1#dirmem#long" dir=8 bytes long = 8
bytes
TEXTO = "A#"

Me.UDPDEBUGGER.RemotePort = 5000
Me.UDPDEBUGGER.RemoteHost = "192.168.1.20"

Me.UDPDEBUGGER.SendData (TEXTO)
End Sub

Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
Me.UDP.SendData "#" + Chr$(KeyCode)
End Sub

Private Sub Form_Load()
Me.UDPDEBUGGER.RemoteHost = "192.168.1.20"
UDP.RemoteHost = "192.168.1.20"
UDP.Bind 3000
UDPDEBUGGER.Bind 5000
' incializar tiempo de los turnos
tiempoJugador1 = 0
tiempoJugador2 = 0

' inicializar datos de la grafica
Me.grafica.Plot.Axis(VtChAxisIdY).ValueScale.Auto = False

' CARGAR COMBO TIPOS DE VARIBALES
Me.MTIPO.AddItem "Byte"
Me.MTIPO.AddItem "Word"
Me.MTIPO.AddItem "Long"
Me.MTIPO.ListIndex = 0

End Sub

Sub ajustaTimer(dato As String)
   tiempo = Val(Mid$(dato, 2, Len(dato) - 1))
   If tiempo >= 0 And tiempo < 20001 Then
      tOut = tiempo
      Me.Timer1.Enabled = True
   End If

End Sub

Private Sub LINEAMENSAJE_Click()
Me.SetFocus
End Sub
```

```
Private Sub LINEAMENSAJE_GotFocus()
Me.Command1.SetFocus
End Sub

Private Sub LINEAMENSAJE_KeyPress(KeyAscii As Integer)
Me.UDP.SendData "#" + Chr$(KeyAscii)
KeyAscii = 0
End Sub

Private Sub mactdepuracion_Click()
Select Case mactdepuracion.Checked
   Case True
      mactdepuracion.Checked = False
           For a = 1 To 10
        Me.etiquetaDepuracion(a).Visible = False
     Next
      Me.ascii.Visible = False
      Me.dir.Visible = False
      Me.hexa.Visible = False
      Me.bloquerefresh.Visible = False
      Me.variablerefresh = False
      Me.variable.Visible = False
      Me.tipo.Visible = False
      Me.valorvariable.Visible = False
      Me.memini.Visible = False
      Me.tamaño.Visible = False
      Me.variablerefresh.Visible = False
      Me.DIRECCION.Visible = False
      Me.MTIPO.Visible = False
      Me.PARAR.Visible = False
    Case False
     mactdepuracion.Checked = True
      Call leeFicheroConf
     For a = 1 To 10
        Me.etiquetaDepuracion(a).Visible = True
     Next
      Me.ascii.Visible = True
      Me.dir.Visible = True
      Me.hexa.Visible = True
      Me.bloquerefresh.Visible = True
      Me.variablerefresh = True
      Me.variable.Visible = True
      Me.tipo.Visible = True
      Me.valorvariable.Visible = True
      Me.memini.Visible = True
      Me.tamaño.Visible = True
      Me.variablerefresh.Visible = True
      Me.PARAR.Visible = True
      Me.DIRECCION.Visible = True
      Me.MTIPO.Visible = True
```

```
End Select
End Sub

Private Sub mmostrarudp_Click()
  Select Case mmostrarudp.Checked
     Case True
        mmostrarudp.Checked = False
        Label1.Visible = False
        Text1.Visible = False

    Case False
        mmostrarudp.Checked = True
        Label1.Visible = True
        Text1.Visible = True

  End Select

End Sub


Private Sub mvisgrafica_Click()
Select Case mvisgrafica.Checked
   Case True
       mvisgrafica.Checked = False
       Me.grafica.Visible = False
   Case False
       mvisgrafica.Checked = True
       Me.grafica.Visible = True
End Select
End Sub

Private Sub PARAR_Click()
' formato envio peticiones  "A#"
TEXTO = "A#"
Me.UDPDEBUGGER.RemotePort = 5000
Me.UDPDEBUGGER.RemoteHost = "192.168.1.20"
Me.UDPDEBUGGER.SendData (TEXTO)
End Sub

Private Sub Timer1_Timer()
If tOut > Timer1.Interval Then
   tOut = tOut - Timer1.Interval
Else
   tOut = 0
   Me.Timer1.Enabled = False
```

```
End If

' mostrar tiempo restante
Me.tiempo.Caption = Format$(tOut / 1000, "00.0#")
If tOut < 5000 Then
   Me.tiempo.ForeColor = &HFF&       ' rojo
   Me.TRESTA.BackColor = &HFF&       ' rojo
Else
   Me.tiempo.ForeColor = &H0&        ' negro
   Me.TRESTA.BackColor = &H0&        ' negro
End If

' actualizar barra de tiempos
   Me.TRESTA.Width = 11295 * (tOut / 20000)
   Me.TRESTA.Left = 2040 + (11295 - Me.TRESTA.Width)

' ajustar tiempo de juego de cada jugador
If tiempoJugador1 > tiempoJugador2 Then
   t = tiempoJugador1
Else
   t = tiempoJugador2
End If

Me.grafica.Plot.Axis(VtChAxisIdY).ValueScale.Maximum = 20000 * (Int(t
/ 20000) + 1)

   If turnoActual = 1 Then
        tiempoJugador1 = tiempoJugador1 + Timer1.Interval
        Me.grafica.Column = 1
        Me.grafica.Data = tiempoJugador1
   Else
        tiempoJugador2 = tiempoJugador2 + Timer1.Interval
        Me.grafica.Column = 2
        Me.grafica.Data = tiempoJugador2

   End If

End Sub

Private Sub UDP_DataArrival(ByVal bytesTotal As Long)
Dim dato As String
On Error Resume Next
UDP.GetData dato
Me.Text1.Text = Time$ + "  " + dato
Call mensaje(dato)

End Sub
Private Sub UDP_Error(ByVal Number As Integer, Description As String,
ByVal Scode As Long, ByVal Source As String, ByVal HelpFile As
String, ByVal HelpContext As Long, CancelDisplay As Boolean)
On Error Resume Next
```

```
Resume Next
End Sub

Private Sub UDPDEBUGGER_DataArrival(ByVal bytesTotal As Long)
Dim dato As String
'On Error Resume Next
UDPDEBUGGER.GetData dato
Me.Text1.Text = Time$ + "  " + dato
Call mensajeDebugger(dato)

End Sub

Private Sub variable_Click()
Combo2.ListIndex = variable.ListIndex
Combo3.ListIndex = variable.ListIndex
Combo4.ListIndex = variable.ListIndex
DIRECCION.Caption = UCase(Combo4.Text)
tipo.Caption = Combo2.Text
Me.valorvariable.Caption = "?????????"
End Sub



Private Sub variablerefresh_Click()
' formato envio peticiones  "1#dirmem#long" dir=8 bytes long = 8
bytes
TEXTO = "1#" & Me.DIRECCION.Caption & "#"
Select Case Me.MTIPO.ListIndex
      Case 0
        ' byte
        TEXTO = TEXTO & "00000001"
      Case 1
        ' word
        TEXTO = TEXTO & "00000002"
      Case Else
        ' word
        TEXTO = TEXTO & "00000004"
End Select

Me.UDPDEBUGGER.RemotePort = 5000
Me.UDPDEBUGGER.RemoteHost = "192.168.1.20"

Me.UDPDEBUGGER.SendData (TEXTO)

End Sub

Private Sub X_MouseDown(Index As Integer, Button As Integer, Shift As
Integer, X As Single, Y As Single)
Dim tecla As String
' para controlar el movimiento del control
tmpX = X
```

```
tmpY = Y

' enviar ficha para mover
Select Case Me.X(Index).Top
     Case 480
        Select Case Me.X(Index).Left
           Case 3360
              tecla = "1"
           Case 6960
              tecla = "2"
           Case 10320
              tecla = "3"
        End Select
     Case 3840
        Select Case Me.X(Index).Left
           Case 3360
              tecla = "4"
           Case 6960
              tecla = "5"
           Case 10320
              tecla = "6"
        End Select
     Case 7320
        Select Case Me.X(Index).Left
           Case 3360
              tecla = "7"
           Case 6960
              tecla = "8"
           Case 10320
              tecla = "9"
        End Select
End Select
Me.UDP.SendData ("#" + tecla)
' Deshabilitar recepcion mensajes tablero mientras arrastro ficha
para filtrar mensajes tablero con ficha quitada
desplazandoFicha = True

End Sub

Private Sub X_MouseMove(Index As Integer, Button As Integer, Shift As
Integer, X As Single, Y As Single)
If desplazandoFicha = False Then Exit Sub
If Button = 1 Then
DoEvents
Me.X(Index).Left = Me.X(Index).Left + (X - tmpX)
Me.X(Index).Top = Me.X(Index).Top + (Y - tmpY)
End If
End Sub

Private Sub X_MouseUp(Index As Integer, Button As Integer, Shift As
Integer, X As Single, Y As Single)
```

```
Dim tecla As String

Select Case Me.X(Index).Top
    Case Is < 2160:
       ' primera fila
       temp = 480
    Case Is < 5580
       temp = 3840
    Case Else
       temp = 7320
End Select
   Me.X(Index).Top = temp

       Select Case Me.X(Index).Left
    Case Is < 5160
       ' primera columna
       temp = 3360
    Case Is < 8640
       temp = 6960
    Case Else
       temp = 10320
End Select
   Me.X(Index).Left = temp

' retornar celda destino
Select Case Me.X(Index).Top
     Case 480
        Select Case Me.X(Index).Left
           Case 3360
              tecla = "1"
           Case 6960
              tecla = "2"
           Case 10320
              tecla = "3"
        End Select
     Case 3840
        Select Case Me.X(Index).Left
           Case 3360
              tecla = "4"
           Case 6960
              tecla = "5"
           Case 10320
              tecla = "6"
        End Select
     Case 7320
        Select Case Me.X(Index).Left
           Case 3360
              tecla = "7"
           Case 6960
              tecla = "8"
           Case 10320
```

```
              tecla = "9"
        End Select
End Select
' anular desplazamiento de ficha para filtra mensajes tablero
desplazandoFicha = False
Me.UDP.SendData ("#" + tecla)
End Sub
```

10.2 CODIGO VISUAL BASIC APLICACIÓN TERMINAL UDP

```
Private Sub Combo1_Click()
Select Case Me.Combo1.ListIndex
   Case 0 ' DESTINO COLDFIRE
      Me.IPDESTINO.Text = "192.168.1.20"
      Me.PUERTODESTINO.Text = "4000"
      Me.PUERTOESCUCHA.Text = "3000"
   Case 1 ' DESTINO TERMINAL 3RAYA
      Me.IPDESTINO.Text = "127.0.0.1"
      Me.PUERTODESTINO.Text = "3000"
      Me.PUERTOESCUCHA.Text = "4000"
End Select
Me.PUERTOESCUCHAD.Text = 5000
Me.PUERTODESTINODEBUG = 5000
Me.UDP.Close
Me.UDP.Bind Me.PUERTOESCUCHA.Text
'Me.UDPDEBUGGER.Close
'Me.UDPDEBUGGER.Bind Me.PUERTOESCUCHAD.Text
End Sub

Private Sub Command1_Click()
UDP.RemoteHost = Me.IPDESTINO.Text
UDP.RemotePort = 7000 'Me.PUERTODESTINO.Text  ' puerto deSTINO
UDP.SendData "1" + Me.ENVIARTABLERO.Text
End Sub


Private Sub Command2_Click()
UDP.RemoteHost = Me.IPDESTINO.Text
UDP.RemotePort = Me.PUERTODESTINO.Text  ' puerto deSTINO
UDP.SendData "2" + Me.ENVIARTIEMPO.Text
End Sub

Private Sub Command3_Click()
UDP.RemoteHost = Me.IPDESTINO.Text
UDP.RemotePort = Me.PUERTODESTINO.Text  ' puerto deSTINO JUEGO

UDP.SendData "0" + Me.ENVIARMENSAJE.Text
End Sub

Private Sub Command4_Click()
Me.RECIBIDO.Text = ""
End Sub

Private Sub Command5_Click()
UDP.RemoteHost = Me.IPDESTINO.Text
UDP.RemotePort = Me.PUERTODESTINO.Text  ' puerto deSTINO
UDP.SendData "3" + Me.ENVIARTURNO.Text
```

```
End Sub

Private Sub Command6_Click()
UDPDEBUGGER.RemoteHost = Me.IPDESTINO.Text
UDPDEBUGGER.RemotePort = Me.PUERTODESTINODEBUG   ' puerto deSTINO
UDPDEBUGGER.SendData "1" + Me.VARIABLE.Text
End Sub

Private Sub Command7_Click()
UDPDEBUGGER.RemoteHost = Me.IPDESTINO.Text
UDPDEBUGGER.RemotePort = Me.PUERTODESTINODEBUG   ' puerto deSTINO
UDPDEBUGGER.SendData Me.BLOQUE.Text
End Sub

Private Sub Form_Load()
Combo1.Clear
Combo1.AddItem "Coldfire"
Combo1.AddItem "Terminal 3 en raya"
Combo1.ListIndex = 1

End Sub

Private Sub UDP_DataArrival(ByVal bytesTotal As Long)
Dim dato As String
On Error Resume Next

UDP.GetData dato
Me.RECIBIDO.Text = Me.RECIBIDO.Text + Chr$(13) + Chr$(10) + Time$ +
">  " + dato
Me.RECIBIDO.SelStart = Len(Me.RECIBIDO.Text)
End Sub

Private Sub UDP_Error(ByVal Number As Integer, Description As String,
ByVal Scode As Long, ByVal Source As String, ByVal HelpFile As
String, ByVal HelpContext As Long, CancelDisplay As Boolean)
On Error Resume Next
End Sub
```

10.3 CODIGO APLICACIÓN 3 EN RAYA (COLDFIRE) 3RAYA.LIB

```
/*   JUEGO TRES EN RAYA DESARROLLADO POR
        FERNANDO NUÑEZ SERRANO
        ALEJANDRO BADOLATO MARTIN
        ver. 28-05-2008 10:56  // codigo definitivo
*/

#include "m5272lib.c"
#include "m5272gpio.c"
#include "clienteUdp.c"

#define NUM_FILAS 4 //tetiempoTurnoclado matricial
#define NUM_COLS 4 //teclado matricial
#define EXCIT 1
#define RETARDOTECLA 1150
#define UNJUGADOR 1
#define DOSJUGADORES 2
#define NULO 0 //valor de retorno nulo
#define MAXFILAS 3//filas tablero
#define MAXCOLS 3 //columnas tablero
#define MAXFICHAS 9 //max numero fichas en tablero
#define VACIO '_' //celda tablero vacia
#define FICHAS1 'X' // fichas con las que juega el jugador uno
#define FICHAS2 'O' // fichas con las que juega la maquina o player2
#define NUMIMPORTANCIAS 6 //distintos tipos de importancias (mejor
jugada automatica)
#define GANA 5 // importancia de ganar
#define EVITAPERDER 4// importancia evitar perder en la proxima
tirada
#define SIPUEDEGANARESQUINA 3// importancia de si poder ganar en
proximo turno insertando en una esquina(mejor)
#define SIPUEDEGANAR 2// importancia de si poder ganar en proximo
turno
#define NOPUEDEGANARESQUINA 1// importancia de no poder ganar en
proximo turno insertando en una esquina(mejor)
#define NOPUEDEGANAR 0// importancia de no poder ganar en proximo
turno
#define MAX_TURNO 20000// maximo tiempo ms. turno de cada jugador
#define MAX_TONOS_MELODIA  51// num max de notas de la melodia de
fondo con pausa final
#define MAX_TONOS_GANADOR  4// num max de notas de la melodia de
ganador
#define MAX_TONOS_PERDEDOR  4// num max de notas de la melodia de
ganador
//#define FACTOR_TMR0 1536 //  =(decalado total del contador) con
TMR0=$2F2C
```

```
// declaración de variables
BYTE tecla='A'; // tecla capturada del teclado
WORD tipoJuego=0; //(=1 con la máquina . =2 con otro jugador)
BYTE tablero[MAXFILAS][MAXCOLS]; //tablero de partida
WORD numeroFichas; //numero fichas en el tablero
WORD numeroFichasMaquina; //
WORD numFila,numColumna; //fila y columna seleccionada
WORD turnoActual; // nº de jugador que tiene su turno
WORD tiempoTurno; //nº de ms. desde inicio del turno
BOOL finTurno=FALSE; //flag para interrumpir el proceso normal de un
turno (Tout20 seg)
// comunes para las 3 melodias
WORD notaActual; // puntero de melodia que marca frecuencia de la
nota que se está reproduciendo
WORD duracionNotaActual; // tiempo que la nota lleva reproduciendose
(ms)
WORD duracionTotalNotaActual; // puntero que marca la duración total
de la nota que debe reproducirse (ms)

// flags
BOOL suenaPerdedor; //=true se reproduce melodia perdedor
BOOL suenaGanador;
BOOL suenaMelodia; //=true se reproduce melodia fondo
BOOL mensajeUdpRx; //= true cuando se ha recibido un mensaje


// consatantes
static int puertoDestino3R = 3000; // puerto destino para envio UDP
juego 3raya
static int puertoEscucha3R=4000; // puerto en el que escuchamos
peticiones juego 3raya
static int puertoEscuchaDEBUG=5000; // puerto en el que escuchamos
peticiones DEBUGGER UDP

// TRR0= MCF_OSC/(FACTOR_TMR0*frec nota) (en Hz)
//LONG static TMR0_CONF=0x2F2C; // $2F2C RST=0 CLK=10 FRR=1 ORI=0
OM=1 CE=00PS=$2F=47 (CONFIGURADO)
LONG static TMR0_ON=0x2F2D; // $2F2D  RST=1 CLK=10 FRR=1 ORI=0 OM=1
CE=00 PS=$2F=47 (FUNCIONANDO)
LONG static TMR0_OFF=0x2F2C; // $2F2C RST=0 CLK=10 FRR=1 ORI=0 OM=1
CE=00 PS=$2F=47 (PARADO Y CONFIGURADO)
LONG static TMR1_CONF=0x181C; // RST=0 CLK=10 FRR=1 ORI=1 OM=0 CE=00
PS=$18=24 (CONFIGURADO)
LONG static TMR1_ON=0x181D; // RST=1 CLK=10 FRR=1 ORI=1 OM=0 CE=00
PS=$18=24 (FUNCIONANDO)
LONG static TRR1_CONF=0xA5; // $A5=165 cuentas de TMR1 = 1 mseg.
(EXACTO!!)
//WORD static FACTORTMR0=1536; //  =(decalado total del contador) con
TMR0=$2F2C
// frecs de las notas de la melodia de fondo
```

```
WORD static
melodia[MAX_TONOS_MELODIA]={784,784,784,1047,1047,1568,0,1568,1760,15
68,1397,1568,0,784,784
,784,1047,1047,1568,0,1568,1760,1568,1397,1760,1568,0,1047,1047,1047,
1976,1568,1047
,1047,1047,1976,1568,1047,1047,1047,1976,1760,1976,2093,1047,1047,104
7,1047,1047
,1047,0};
//duracion de cada nota de melodia de fondo
WORD static
tiempoMelodia[MAX_TONOS_MELODIA]={166,166,166,333,166,666,166,166,111
,83,166,1333,166,166,166
,166,333,166,666,166,166,111,83,166,166,444,333,166,166,166,444,222,1
66,166,166
,444,222,166,166,166,166,166,166,666,166,166,166,166,166,444,600};
WORD static melodiaGanador[4]={523,659,784,0}; //frecs de las notas
de la melodia del ganador
WORD static timeMelodiaGanador[4]={300,300,300,1000}; //duracion de
las notas de la melodia del ganador
WORD static melodiaPerdedor[4]={392,330,262,0}; //frecs de las notas
de la melodia del perdedor
WORD static timeMelodiaPerdedor[4]={500,500,500,1000}; //duracion de
las notas de la melodia del perdedor
WORD static FACTOR_TMR0=1536; // decalado total del contador con
TMR0=$2F2C
```

```
//declaracion de funciones. (para no tener que ponerlas por orden de
llamada)
BOOL colocarFicha(WORD importancia);
BOOL compruebaVictoria(void);
BOOL desplazar (void);
BOOL esCeldaCentral(WORD fila,WORD columna);
BOOL esCeldaVacia(WORD fila,WORD columna);
BOOL esCeldaValida(WORD fila,WORD columna);
BOOL esEsquina(WORD fila,WORD columna);
BOOL esMejorJugada (WORD mejorValor,WORD jugadaActual);
BOOL insertarFicha(WORD fila,WORD columna);
BOOL quitarFicha(WORD fila,WORD columna);
BOOL quitarFichaValido(WORD fila,WORD columna);
BOOL victoriaEnColumna(void);
BOOL victoriaEnDiagonal(void);
BOOL victoriaEnFila(void);
BYTE contadorColumna(WORD columna);
BYTE simbolo(WORD jugador);
BYTE teclado(void);
void __init(void);
void ajustaTimer0(WORD trr);
void ajustaTimer1(void);
void bucleMain(void);
void cambiaJugador (void);
void cambiaJugadorUdp(void);
void cambiaNota(void);
void cambiaNotaGanador(void);
void cambiaNotaPerdedor(void);
void copiarArray(BYTE leerArray[MAXFILAS][MAXCOLS],BYTE copiarArray
[MAXFILAS][MAXCOLS]);
void iniciaMelodia(void);
void iniciaMelodiaGanador(void);
void iniciaMelodiaPerdedor(void);
void iniciaTablero(void);
void jugadaJugador(void);
void jugadaMaquina(void);
void jugarPartida(void);
void jugarTurno(void);
void leerCelda(void);
void mostrarMenu(void);
void muestraTablero(void);
void rutina_int1(void);
void rutina_int2(void);
void rutina_int3(void);
void rutina_int4(void);
void rutina_tout0(void);
void rutina_tout1(void);
void rutina_tout2(void);
void rutina_tout3(void);
void selTipoJuego(BYTE tecla);
```

```
void toutGanador(void);
void toutMelodia(void);
void toutPerdedor(void);
void muestraTableroUdp(void);
WORD buscaMejorOpcion (WORD fila,WORD columna);
WORD calculaPeriodo(WORD frecuenciaNota);
WORD comprobarColumna(WORD columna,WORD fila);
WORD comprobarDiagonalDerecha(void);
WORD comprobarDiagonalIzda(void);
WORD comprobarFila(WORD fila,WORD col);
WORD contadorDiagonalDcha(void);
WORD contadorDiagonalIzda(void);
WORD contadorFila(WORD fila);
WORD jugadaAutomatica (void);
void jugadaAutomaticaDesplaza(void);
WORD jugadaContrario(void);
BOOL enDiagonalDcha(WORD fila,WORD columna);
BOOL enDiagonalIzda(WORD fila,WORD columna);
BOOL FichaSeMueve (int fila,int columna);
```

```
//-----------------------------------------------------------
//   BOOL enDiagonalIzda(WORD fila,WORD columna)
//   comprueba si la celda está en la diagonal izquierda
//-----------------------------------------------------------
BOOL enDiagonalIzda(WORD fila,WORD columna){
    BOOL temp=FALSE;
    if ((fila==0) & (columna==0)) temp=TRUE;
    if ((fila==2) & (columna==2)) temp=TRUE;
    return temp;
}
```

```
//-----------------------------------------------------------
//   BOOL enDiagonalDcha(WORD fila,WORD columna)
//   comprueba si la celda está en la diagonal derecha
//-----------------------------------------------------------
BOOL enDiagonalDcha(WORD fila,WORD columna){
    BOOL temp=FALSE;
    if ((fila==0) & (columna==2)) temp=TRUE;
    if ((fila==2) & (columna==0)) temp=TRUE;
    return temp;
}
```

```
//--------------------------------------------------------
// BYTE teclado(void)
//
//   Explora el teclado matricial y devuelve la tecla
//   pulsada
//--------------------------------------------------------
BYTE teclado(void)
{
  BYTE tecla;
  BYTE fila, columna, fila_mask;
  static char teclas[NUM_FILAS][NUM_COLS] = {{"123C"},
                              {"456D"},
                              {"789E"},
                              {"A0BF"}};
  // Bucle de exploración del teclado
  while(finTurno==FALSE){
     // Excitamos una columna
    for(columna = NUM_COLS - 1; columna >= 0; columna--){
      set_puertoS(EXCIT << columna);          // Se envía la
excitación de columna
      retardo(RETARDOTECLA);                  // Esperamos respuesta
de optoacopladores
      // Exploramos las filas en busca de respuesta
      for(fila = NUM_COLS - 1; fila >= NULO; fila--){
        fila_mask = EXCIT << fila;  // Máscara para leer el bit de la
fila actual
        if(lee_puertoE() & fila_mask){   // Si encuentra tecla
pulsada,
          while(lee_puertoE() & fila_mask);//   Esperamos a que se
suelte
          retardo(RETARDOTECLA);         // Retardo antirrebotes
          tecla=teclas[fila][columna];// Devolvemos la tecla pulsada
          outch(tecla); //eco pantalla
          output("\r\n");
          return tecla;
        }
      }
      // Siguiente columna
    }
   // Exploración finalizada sin encontrar una tecla pulsada
    if (mensajeUdpRx==TRUE){
        mensajeUdpRx=FALSE;
        // comprobar que tiene el formato correcto
        if (bufferUdpRX[0]=='#'){
           return bufferUdpRX[1];
        }
    }
  } // while
   //retornamos algo por defecto si t_out
   return 'A';
}
```

```
//--------------------------------------------------------
// void mostrarTablero(void)
//
// muestra el tablero
//
//--------------------------------------------------------
void muestraTablero(void){
     WORD i,j;
     WORD  k=0; //numero de caracteres de cada linea
     BYTE mensaje[5];



     for (i=0;i < MAXFILAS;i++){
     k=0;
         for (j=0;j < MAXCOLS;j++){
             mensaje[k]=tablero[i][j];
             k++;
             mensaje[k]=' ';
             k++;
         }
         output(mensaje);
         output("\r\n");


     }
     // Mostrar tablero via UDP
     muestraTableroUdp();
     return;
}
```

```
//-------------------------------------------------------
// void muestraTableroUdp(void)
//
// envia el tablero por UDP
//
//-------------------------------------------------------
void muestraTableroUdp(void){
   int i,j,k;
   char array[12];
   array[0]='1'; // codigo de envio de tablero
   // recorrer el tablero
   k=1;
   for (i=0;i < MAXFILAS;i++){
      for (j=0;j < MAXCOLS;j++){
             array[k]=tablero[i][j];
             k++;
      }
   }
   // poner fin de la cadena
   array[k]='#';
   k++;
   array[k]='#';
   enviaUdp(&array[0],puertoDestino3R);


}
```

```
//-------------------------------------------------------
// void iniciaTablero(void)
//
// inicializa el tablero
//
//-------------------------------------------------------
void iniciaTablero(void){
     WORD i,j;
     numeroFichas=0;
     numeroFichasMaquina=0;
     for (i=0;i < MAXFILAS;i++){
         for (j=0;j < MAXCOLS;j++){
             tablero[i][j]=VACIO;
         }
     }
 return;
}
```

```
//-----------------------------------------------------------
//   void cambiaJugador (void)
//
//
//   cambia el turno actual del jugador
//
//-----------------------------------------------------------
void cambiaJugador (void){
      if (turnoActual==1){
        turnoActual=2;
      } else turnoActual=1;
return;
}





//-----------------------------------------------------------
//   void cambiaJugadorUdp (void)
//
//
//   cambia el turno actual del jugador actual y envia
//   cambio al cliente udp
//
//-----------------------------------------------------------
void cambiaJugadorUdp (void){
      if (turnoActual==1){
        turnoActual=2;
        enviaUdp("32##",puertoDestino3R);

      } else {
        turnoActual=1;
        enviaUdp("31##",puertoDestino3R);
      }
return;
}


//--------------------------------------------------------
// BYTE simbolo(WORD jugador)
//
// retorna  ficha del jugador
//
//--------------------------------------------------------
BYTE simbolo(WORD jugador){
    if (jugador==1){
       return FICHAS1;
    }else  return FICHAS2;
}
```

```
//-------------------------------------------------------
// BOOL esEsquina(WORD fila,WORD columna)
//
// retorna  TRUE si es celda esquina
//
//-------------------------------------------------------
BOOL esEsquina(WORD fila,WORD columna){
    // comprobar esqunas superiores
    if (fila==0 && (columna==0||columna==2)){
       return TRUE;
    }
    // comprobar esquinas inferiores
    if (fila==2 && (columna==0||columna==2)){
       return TRUE;
    }
    return FALSE;
}
```

```
//-------------------------------------------------------
// BYTE contadorColumna(WORD fila)
//
// retorna  nº de fichas del turnoActual en la columna
//
//-------------------------------------------------------
BYTE contadorColumna(WORD columna)    {
     WORD contador=0; // cuenta las fichas del jugador en el segmento
a analizar
     WORD j;
        // contar las fichas en esa fila
         for(j=0;j<MAXFILAS;j++){
              if (tablero[j][columna]==simbolo(turnoActual)){
                 contador++;
              }
         }
         return contador;
     }
```

```
//--------------------------------------------------------
// WORD contadorFila(WORD fila)
//
// retorna  nº de fichas del turnoActual en la fila
//
//--------------------------------------------------------
WORD contadorFila(WORD fila)    {
     WORD contador=0; // cuenta las fichas del jugador en el segmento
a analizar
     WORD j;
        // contar las fichas en esa fila
         for(j=0;j<MAXCOLS;j++){
              if (tablero[fila][j]==simbolo(turnoActual)){
                 contador++;
              }
         }
         return contador;
     }




//--------------------------------------------------------
// WORD contadorDiagonalIzda(void)
//
// retorna  nº de fichas del turnoActual en la diagonal izquierda
//
//--------------------------------------------------------
WORD contadorDiagonalIzda(void)    {
     WORD contador=0; // cuenta las fichas del jugador en el segmento
a analizar
    if (tablero[0][0]==simbolo(turnoActual)) contador++;
    if (tablero[1][1]==simbolo(turnoActual)) contador++;
    if (tablero[2][2]==simbolo(turnoActual)) contador++;
    return contador;
}
```

```
//-------------------------------------------------------
// WORD contadorDiagonalDcha(void)
//
// retorna  nº de fichas del turnoActual en la diagonal derecha
//
//-------------------------------------------------------
WORD contadorDiagonalDcha(void)    {
     WORD contador=0; // cuenta las fichas del jugador en el segmento
a analizar
    if (tablero[0][2]==simbolo(turnoActual)) contador++;
    if (tablero[1][1]==simbolo(turnoActual)) contador++;
    if (tablero[2][0]==simbolo(turnoActual)) contador++;
         return contador;
}
```

```
//-------------------------------------------------------
// BOOL victoriaEnFila(void)
//
// retorna  TRUE si el turnoActual ha ganado por filas
//
//-------------------------------------------------------
BOOL victoriaEnFila(void)    {
     WORD i;
     // mirar victoria por 3enraya en alguna fila
     for(i=0;i<MAXFILAS;i++){
         // contar las fichas en esa fila y ver si hay 3
        if (contadorFila(i)==3) return TRUE;
     }
     // como no se detecto victoria en filas develve FALSE
     return FALSE;
}
```

```
//-------------------------------------------------------
// BOOL victoriaEnColumna(void)
//
// retorna  TRUE si el turnoActual ha ganado por columnas
//
//-------------------------------------------------------
BOOL victoriaEnColumna(void)    {
     WORD j;
     // mirar victoria por 3enraya en alguna columna
     for(j=0;j<MAXCOLS;j++){
         // contar las fichas en esa columna y ver si hay 3
         if (contadorColumna(j)==3) return TRUE;
     }
     // como no se detecto victoria en columnas develve FALSE
     return FALSE;
}
```

```
//-------------------------------------------------------
// BOOL victoriaEnDiagonal(void)
//
// retorna  TRUE si el turnoActual ha ganado por diagonales
//
//-------------------------------------------------------
BOOL victoriaEnDiagonal(void)    {
     // comprobar diagonal IZDA
     if (contadorDiagonalIzda()==3) return TRUE;
     // comprobar diagonal DCHA
     if (contadorDiagonalDcha()==3) return TRUE;
     return FALSE;
}
```

```
//-------------------------------------------------------
// BOOL compruebaVictoria(void)
//
// retorna  TRUE si el turnoActual ha ganado la partida
//
//-------------------------------------------------------
BOOL compruebaVictoria(void){
     if (victoriaEnFila()){
             enviaUdp("20##",puertoDestino3R); // enviar tiempo de
turno restante
             return TRUE;

     }
     if (victoriaEnColumna()){
             enviaUdp("20##",puertoDestino3R); // enviar tiempo de
turno restante
             return TRUE;
     }
     if (victoriaEnDiagonal()){
             enviaUdp("20##",puertoDestino3R); // enviar tiempo de
turno restante
             return TRUE;
     }
     return FALSE;
}
```

```
//-------------------------------------------------------
// BOOL esCeldaCentral(WORD fila,WORD columna)
//
// retorna  TRUE si es la celda central
//
//-------------------------------------------------------
BOOL esCeldaCentral(WORD fila,WORD columna){
    if (fila==1 && columna==1){
       return TRUE;
    }else  return FALSE;
}
```

```
//-------------------------------------------------------
// BOOL esCeldaValida(WORD fila,WORD columna)
//
// comprobar que la celda está en limites
// retorna  TRUE si es una celda del tablero
//
//-------------------------------------------------------
BOOL esCeldaValida(WORD fila,WORD columna){
    // comprobar que la celda está en limites
    if (fila>=0 && columna>=0 && fila<3 && columna<3){
       return TRUE;
    }else return FALSE;
}
```

```
//-------------------------------------------------------
// BOOL esCeldaVacia(WORD fila,WORD columna)
//
// retorna  TRUE si es una celda vacia
//
//-------------------------------------------------------
BOOL esCeldaVacia(WORD fila,WORD columna){
       //comprobar que la celda está vacia
       if (tablero[fila][columna]==VACIO){
          return TRUE;
       }else return FALSE;
}
```

```
//-------------------------------------------------------
// WORD comprobarfila(WORD fila,WORD columna)
//
// retorna  importancia de la casilla según la fila con
// el siguiente orden de prioridad
// GANA
// EVITAPERDER
// SIPUEDEGANARESQUINA
// SIPUEDEGANAR
// NOPUEDEGANARESQUINA
// NOPUEDEGANAR
//-------------------------------------------------------
WORD comprobarFila(WORD fila,WORD columna){
     WORD mias,contrario;
     mias=contadorFila(fila);
     cambiaJugador();
     contrario=contadorFila(fila);
     // volver a dejar el mismo jugador
     cambiaJugador();
     if ((mias==2) && (contrario==0)) return GANA;
     if ((contrario==2) && (mias==0)) return EVITAPERDER;
     if (contrario==0 && mias==1){
         if (esEsquina(fila,columna)==TRUE){
             return SIPUEDEGANARESQUINA;
         }else return SIPUEDEGANAR;
     }
     // resto de los casos
     if (esEsquina(fila,columna)==TRUE){
         return NOPUEDEGANARESQUINA;
     }

     return NOPUEDEGANAR;



}
```

```
//------------------------------------------------------
// WORD comprobarColumna(WORD fila,WORD columna)
//
// retorna  importancia de la casilla según la columna con
// el siguiente orden de prioridad
// GANA
// EVITAPERDER
// SIPUEDEGANARESQUINA
// SIPUEDEGANAR
// NOPUEDEGANARESQUINA
// NOPUEDEGANAR
//------------------------------------------------------
WORD comprobarColumna(WORD fila,WORD columna){
     WORD mias,contrario;
     mias=contadorColumna(columna);
     cambiaJugador();
     contrario=contadorColumna(columna);
     // volver a dejar el mismo jugador
     cambiaJugador();
     if ((mias==2) && (contrario==0)) return GANA;
     if ((contrario==2) && (mias==0)) return EVITAPERDER;
     if (contrario==0 && mias==1){
      if (esEsquina(fila,columna)==TRUE){
             return SIPUEDEGANARESQUINA;
         }else return SIPUEDEGANAR;
     }

     // resto de los casos
     if (esEsquina(fila,columna)==TRUE){
         return NOPUEDEGANARESQUINA;
     }

     return NOPUEDEGANAR;
}
```

```
//--------------------------------------------------------
// WORD comprobarDiagonalIzda(void)
//
// retorna  importancia de la casilla según la diagonal izquierda con
// el siguiente orden de prioridad
// GANA
// EVITAPERDER
// SIPUEDEGANARESQUINA
// SIPUEDEGANAR
// NOPUEDEGANARESQUINA
// NOPUEDEGANAR
//--------------------------------------------------------
WORD comprobarDiagonalIzda(void){
     WORD mias,contrario;
     mias=contadorDiagonalIzda();
     cambiaJugador();
     contrario=contadorDiagonalIzda();
     // volver a dejar el mismo jugador
     cambiaJugador();
     if ((mias==2) && (contrario==0)) return GANA;
     if ((contrario==2) && (mias==0)) return EVITAPERDER;
     if (contrario==0 && mias==1) return SIPUEDEGANARESQUINA;
     // resto de los casos
     return NOPUEDEGANARESQUINA;
}
```

```
//-------------------------------------------------------
// WORD comprobarDiagonalDerecha(void)
//
// retorna  importancia de la casilla según la diagonal derecha con
// el siguiente orden de prioridad
// GANA
// EVITAPERDER
// SIPUEDEGANARESQUINA
// SIPUEDEGANAR
// NOPUEDEGANARESQUINA
// NOPUEDEGANAR
//-------------------------------------------------------
WORD comprobarDiagonalDerecha(void){
     WORD mias,contrario;
     mias=contadorDiagonalDcha();
     cambiaJugador();
     contrario=contadorDiagonalDcha();
     // volver a dejar el mismo jugador
     cambiaJugador();
     if ((mias==2) && (contrario==0)) return GANA;
     if ((contrario==2) && (mias==0)) return EVITAPERDER;
     if (contrario==0 && mias==1) return SIPUEDEGANARESQUINA;
     // resto de los casos
     return NOPUEDEGANARESQUINA;
}
```

```
//--------------------------------------------------------
// BOOL insertarFicha(WORD fila,WORD columna)
//
// retorna codigo operacion. TRUE = insertada correctamente y  FALSE
= no insertada
//
//--------------------------------------------------------
BOOL insertarFicha(WORD fila,WORD columna){

     // comprobar que el primer jugador inserta en el centro
     if (numeroFichas==0){
         if (!esCeldaCentral(fila,columna)) {
             // no se ha puesto en el centro siendo la primera ficha
             return FALSE;
         }
     }
     // comprobar que está en dentro de los limites
      if (!esCeldaValida(fila,columna)){
        return FALSE;
     }
     // comprobar que la celda está vacía
     if (esCeldaVacia(fila,columna)){
        tablero[fila][columna] = simbolo(turnoActual);
        numeroFichas++;
        if (turnoActual==2){
            numeroFichasMaquina++;
        }
        return TRUE;
     }else{
        return FALSE;
     }
}
```

```
//------------------------------------------------------
// BOOL quitarFichaValido(WORD fila,WORD columna)
//
// retorna codigo operacion. TRUE = se puede quitar la ficha
//
//------------------------------------------------------
BOOL quitarFichaValido(WORD fila,WORD columna){
     // comprobar que pertence al tablero
     if (esCeldaValida(fila,columna)){
        //comprobar que la celda contiene una ficha del jugador
        if (tablero[fila][columna]==simbolo(turnoActual)){
           //comprobar que no es la celda central
           if (!esCeldaCentral(fila,columna)){
              return TRUE;
           }
        }
     }
     return FALSE;
}
```

```
//------------------------------------------------------
// BOOL quitarFicha(WORD fila,WORD columna)
//
// quita una ficha del tablero verificando que se quita una ficha
// del jugador que tiene el turnoActual.
// retorna codigo operacion. TRUE = eliminada correctamente y FALSE =
no eliminada
//
//------------------------------------------------------
BOOL quitarFicha(WORD fila,WORD columna){
     if (quitarFichaValido(fila,columna)){
        tablero[fila][columna] = '_';
        numeroFichas--;
        return TRUE;
        }
     return FALSE;
}
```

```
//-----------------------------------------------------------
// void selTipoJuego(BYTE tecla)
//
// Metodo que selecciona tipo de Juego.
// devuelve 1= contra la maquina y 2 = 2 jugadores y 0 si no es
seleccion correcta
//-----------------------------------------------------------

void selTipoJuego(BYTE tecla){

    tipoJuego=NULO;
    if (tecla=='A'){
         tipoJuego=UNJUGADOR;
          output("\r\n Juego contra la maquina. Comienza ud. con
fichas X \r\n");
          enviaUdp("0Juego contra la maquina. Comienza ud. con fichas
X##",puertoDestino3R);

    }
    if (tecla=='B'){
         tipoJuego=DOSJUGADORES;
          output("\r\n Juego entre dos jugadores. Comienza el jugador
X \r\n");
          enviaUdp("0Juego entre dos jugadores. Comienza el jugador
X##",puertoDestino3R);
    }
return;
}
```

```
//-----------------------------------------------------------
//  void leerCelda(void)
//  captura una seleccion de celda
//-----------------------------------------------------------
 void leerCelda(void){
      // bucle hasta seleccion correcta de celda
      do{
          switch (teclado()){
              case '1':
                 numFila=0;
                 numColumna=0;
              break;
           case '2':
                 numFila=0;
                 numColumna=1;
              break;
           case '3':
                 numFila=0;
                 numColumna=2;
              break;
              case '4':
                 numFila=1;
                 numColumna=0;
              break;
           case '5':
                 numFila=1;
                 numColumna=1;
              break;
           case '6':
                 numFila=1;
                 numColumna=2;
              break;
              case '7':
                 numFila=2;
                 numColumna=0;
              break;
           case '8':
                 numFila=2;
                 numColumna=1;
              break;
           case '9':
                 numFila=2;
                 numColumna=2;
              break;
           default:
           // tecla no permitida
           numFila=3;
           numColumna=3;
        }
```

```
         if ((!esCeldaValida(numFila,numColumna))&(finTurno==FALSE)){
              output("Celda incorrecta\r\n");
              enviaUdp("0Celda incorrecta##",puertoDestino3R);
         }
      }while
((!esCeldaValida(numFila,numColumna))&(finTurno==FALSE));

   return;
}
```

```
//-----------------------------------------------------------
//  BOOL desplazar (void)
//
//  comprueba si tiene que desplazar o insertar ficha
//  mirando si hay 6 fichas (o mas...error) en tablero
//
//-----------------------------------------------------------
BOOL desplazar (void){
     if (numeroFichasMaquina>2){
          return TRUE;
     } else return FALSE;
}
```

```
//-----------------------------------------------------------
//  void jugadaJugador(void)
//  jugada de un Jugador
//-----------------------------------------------------------
void jugadaJugador(void){
    BOOL fallo=FALSE;
    BYTE tempTablero[MAXFILAS][MAXCOLS]; //copia seg tablero partida
    WORD tempNumFichas;
    copiarArray(tablero,tempTablero);
    tempNumFichas=numeroFichas;
    do{
       fallo=FALSE;
       if (desplazar()){
          output("Seleccione la ficha a mover.\r\n");
          leerCelda(); // si hay timeout no mostrar resto de mensajes
          if (finTurno==FALSE){
              fallo=!quitarFicha(numFila,numColumna);
              if (!fallo){
                 output("Seleccione donde poner la ficha.\r\n");
                 leerCelda();// si no hay un error insertar ficha
                 fallo=!insertarFicha(numFila,numColumna);
              }
          }
       }else {// insertar ficha
           output("Seleccione donde poner la ficha.\r\n");
           leerCelda();
           fallo=!insertarFicha(numFila,numColumna);
       }
       // comprobar si ha habido fallo
       if(fallo){
          if (finTurno==FALSE){
                output("Jugada Incorrecta. Repita su jugada\r\n");
                enviaUdp("0Jugada Incorrecta. Repita su
jugada##",puertoDestino3R);
                 // dejar el tablero como estaba antes de jugar
                 copiarArray(tempTablero,tablero);
                 numeroFichas=tempNumFichas;
                 muestraTablero();
           }
       }
    }while((fallo) & (finTurno==FALSE));
    if (finTurno==TRUE){
                 output("Ha expirado su tiempo. Pierde el
turno\r\n");
           enviaUdp("0Ha expirado su tiempo. Pierde el
turno##",puertoDestino3R);
                 copiarArray(tempTablero,tablero);
                 numeroFichas=tempNumFichas;
    }
return;
}
```

```
//-----------------------------------------------------------
//  void jugadaMaquina(void)
//  jugada de la Maquina
//-----------------------------------------------------------
void jugadaMaquina(void){
   BYTE tempTablero[MAXFILAS][MAXCOLS];


   WORD importancia; //no se usa
    // no se detecta posible fallo de la maquina
    if (desplazar()){
          // almacenar tablero inicial
          copiarArray(tablero,tempTablero); //hacer copia del tablero
          jugadaAutomaticaDesplaza();
    } else {
       importancia=jugadaAutomatica();
    }
return;
}
```

```
//-----------------------------------------------------------
//  void jugarTurno(void)
//  juega un turno de turnoActual indicando por pantalla quien
//  realiza el movimiento
//-----------------------------------------------------------
 void jugarTurno(void){
     tiempoTurno=0; //Inicializar tiempo del turno
     if (numeroFichas>=1)  enviaUdp("220000##",puertoDestino3R); //
enviar tiempo de turno restante
     finTurno=FALSE; //inicializar flag
     switch(tipoJuego){
         case 1: // 1 jugador contra la maquina
              switch(turnoActual){
                 case 1: // turno del jugador
                    output("\r\n Su turno \r\n");
                    enviaUdp("0Su turno##",puertoDestino3R);
              jugadaJugador();
                    break;
                 case 2: // Turno de la maquina
                    output("\r\nTurno de la maquina \r\n");
                    enviaUdp("0Turno de la
maquina##",puertoDestino3R);
                    jugadaMaquina();
                    break;
              }
              break;
         case 2: //2 jugadores
               if (turnoActual==1){
                   output("\r\n Turno del jugador 1\r\n");
               }else output("\r\n Turno del jugador 2\r\n");


           if (turnoActual==1){
                  enviaUdp("0Turno del jugador 1##",puertoDestino3R);
              }else enviaUdp("0Turno del jugador
2##",puertoDestino3R);


              jugadaJugador();
     }
return;
}
```

```
//----------------------------------------------------------
//  void jugarPartida(void)
//  inicia una partida nueva
//----------------------------------------------------------
void jugarPartida(void){
     BOOL finPartida=FALSE;
     // borrar grafica de tiempos en cliente udp
     enviaUdp("30##",puertoDestino3R);
     turnoActual=1; //comienza siempre el jugador 1
     enviaUdp("31##",puertoDestino3R);
     do{
        // jugar turno del turnoActual
        jugarTurno();
        // comprobar ganador
        finPartida=compruebaVictoria();
        // cambiar jugador
        cambiaJugadorUdp();
        muestraTablero(); // pinta el tablero despues de la jugada
     }while (!finPartida);
    // devolver turno al ganador
    cambiaJugador();
    if (tipoJuego==1){
        if (turnoActual==1){
             output("\r\n Enhorabuena. Has ganado a la máquina\r\n");
          enviaUdp("0Enhorabuena. Has ganado a la
máquina##",puertoDestino3R); //no te lo crees ni tu!!
             iniciaMelodiaGanador();
         }else{
             output("\r\n Lo Siento. Has perdido\r\n");
          enviaUdp("0Lo Siento. Has perdido##",puertoDestino3R);
             iniciaMelodiaPerdedor();
         }
    }else {
        if (turnoActual==1){
             output("\r\n Gana el jugador 1\r\n");
             enviaUdp("0Gana el jugador 1##",puertoDestino3R);
             iniciaMelodiaGanador();
         }else {
             output("\r\n Gana el jugador 2\r\n");
             enviaUdp("0Gana el jugador 2##",puertoDestino3R);
             iniciaMelodiaGanador();
    }
}
return;
}
```

```
//-----------------------------------------------------------
//  void mostrarMenu(void)
//  muestra el menu del juego
//-----------------------------------------------------------
void mostrarMenu(void){
 output("Elija el tipo de juego desdeado: (A) contra la máquina (B)
entre dos Jugadores:");
 enviaUdp("0Elija el tipo de juego desdeado: (A) contra la máquina
(B) entre dos Jugadores:##",puertoDestino3R);
// selecionar tipo juego
do{
   selTipoJuego(teclado());
   if (tipoJuego==0) {
      output("\r\n Selección incorrecta. Repita selección \r\n");
      enviaUdp("0Selección incorrecta. Repita
selección##",puertoDestino3R);
   }

}while(tipoJuego==0);

return;
}
```

```
//-----------------------------------------------------------
// void iniciaMelodia(void)
//  Arranca el timer0 con la primera nota de la melodia
//  timer0 NO INTERRUMPE!!
//-----------------------------------------------------------
void iniciaMelodia(void){
     duracionTotalNotaActual= 0; //puntero
     notaActual=0; //puntero
     duracionNotaActual=0; // tiempo en mseg.
     ajustaTimer0(calculaPeriodo(melodia[notaActual]));
     suenaMelodia=TRUE;
     suenaGanador=FALSE; //parar resto melodias por si todavía se
están
     suenaPerdedor=FALSE; //reproduciendo
}
```

```
//-----------------------------------------------------------
// void iniciaMelodiaGanador(void)
//  Arranca el timer0 con la primera nota de la melodia
//  timer0 NO INTERRUMPE!!
//-----------------------------------------------------------
void iniciaMelodiaGanador(void){
     duracionTotalNotaActual= 0; //puntero
     notaActual=0; //puntero
     duracionNotaActual=0; // tiempo en mseg.
     ajustaTimer0(calculaPeriodo(melodiaGanador[notaActual]));
     suenaGanador=TRUE;
     suenaMelodia=FALSE; //parar melodia fondo
}
```

```
//-----------------------------------------------------------
// void iniciaMelodiaPerdedor(void)
//  Arranca el timer0 con la primera nota de la melodia
//  timer0 NO INTERRUMPE!!
//-----------------------------------------------------------
void iniciaMelodiaPerdedor(void){
     duracionTotalNotaActual= 0; //puntero
     notaActual=0; //puntero
     duracionNotaActual=0; // tiempo en mseg.
     ajustaTimer0(calculaPeriodo(melodiaPerdedor[notaActual]));
     suenaPerdedor=TRUE;
     suenaMelodia=FALSE; //parar melodia fondo
}
```

```
//-----------------------------------------------------------
// void bucleMain(void)
//  metodo inicial de la plataforma ENT
//-----------------------------------------------------------
void bucleMain(void){
    BOOL fin=FALSE;
    BOOL eleccion=FALSE; //comprueba que se eligio o A o B
    ajustaTimer1(); //arranca timer 1mseg
    suenaGanador=FALSE;
    suenaPerdedor=FALSE;
    suenaMelodia=FALSE;
    do{
     // se reproduce melodia fondo con la primera interr. de 1 ms
        // mensaje inicial
     output("**************      Juego de Tres en Raya
***************\r\n");
        enviaUdp("0**************      Juego de Tres en Raya
***************##",puertoDestino3R);
     finTurno=FALSE; //evitar tout turno (no hay tout turno)
        iniciaTablero();
        mostrarMenu();
        muestraTablero();
        muestraTableroUdp();
        jugarPartida();
        // otra partida
        iniciaTablero();
        output("\r\n************* ¿OTRA PARTIDA? A=SI  B=NO
***********\r\n");
        enviaUdp("0************* ¿OTRA PARTIDA? A=SI  B=NO
***********##",puertoDestino3R);
     fin=FALSE;
        eleccion=FALSE;
     do{
         switch (teclado()){
               case 'B':
               fin=TRUE ;
               eleccion=TRUE;
               break;
               case 'A':
                  eleccion=TRUE;
                  fin=FALSE;
                  break;
               default:
                  output ("\r\nEleccion Incorrecta\r\n");
                  enviaUdp("0Eleccion Incorrecta##",puertoDestino3R);
             }
         }while(eleccion==FALSE);
    }while(fin==FALSE);
    exit(0);
}
```

```
//----------------------------------------------------------
//  BOOL esMejorJugada (WOORD mejorValor,WORD  jugadaActual)
//
//  devuelve TRUE si jugadaActual tiene mas puntuación que la
//  almacenada hasta el momento
//----------------------------------------------------------
BOOL esMejorJugada (WORD mejorValor,WORD jugadaActual){
     if (jugadaActual>=mejorValor){
        return TRUE;

     } else return FALSE;
}
```

```
//-----------------------------------------------------------
//  WORD buscaMejorOpcion (WORD fila,WORD columna)
//
//  devuelve la mayor importancia que tiene la celda. Se vigila
//  la fila, la columna y si es el caso la diagonal asociada a esa
//  posicion. Según el siguiente criterio de mas a menos:
//  GANA
//  EVITAPERDER
//  SIPUEDEGANAR (en siguiente turno)
//  NOPUEDEGANAR (en siguiente turno)
//-----------------------------------------------------------
WORD buscaMejorOpcion (WORD fila,WORD columna){
     WORD importanciaCasilla=NOPUEDEGANAR; // de la casilla
     WORD importanciaF=NOPUEDEGANAR; // de su fila
     WORD importanciaC=NOPUEDEGANAR; // de su columna
     WORD diagonalIzda=NOPUEDEGANAR; // diagonal IZDA
     WORD diagonalDcha=NOPUEDEGANAR; // diagonal DCHA
     // calcular importancia de la fila
     importanciaF=comprobarFila(fila,columna);
     // calcular importancia de la columna
     importanciaC=comprobarColumna(fila,columna);
       // comprobar si hay que calcular diagonal izda
     if (enDiagonalIzda(fila,columna)){
          diagonalIzda=comprobarDiagonalIzda();
     }

     if (enDiagonalDcha(fila,columna)){
          diagonalDcha=comprobarDiagonalDerecha();
     }
// quedarse con la mejor puntuacion

    //quedarse con la mejor diagonal
     if (esMejorJugada(diagonalIzda,diagonalDcha)){
          importanciaCasilla=diagonalDcha;
          } else importanciaCasilla=diagonalIzda;
    // comprobar si es mejor la fila
     if (esMejorJugada(importanciaCasilla,importanciaF)){
          importanciaCasilla=importanciaF;
          }
     // comprobar si es mejor la columna
          if (esMejorJugada(importanciaCasilla,importanciaC)){
          importanciaCasilla=importanciaC;
          }
// devolver mejor puntuación
    return importanciaCasilla;
}
```

```
//-----------------------------------------------------------
//   BOOL colocarFichaOrdenador (WORD importancia)
//
//   Este método busca en todas las celdas vacias del tablero
//   una celda con la importancia que se le ha pasado como
//   parametro. Si la encuentra inserta la ficha y finaliza.Si
//   no encuentra ninguna celda con esa importancia devuelte
//   FALSE
//
//-----------------------------------------------------------
BOOL colocarFicha(WORD importancia){
     BOOL insertada=FALSE;
     // recorro el tablero
      WORD i,j;
     for ( i=0;(i < MAXFILAS) && (!insertada);i++){
         for ( j=0;(j < MAXCOLS) && (!insertada);j++){
             if (tablero[i][j]==VACIO){
                 // buscar la maxima importancia para esa celda
                 // si es igual a la buscada se inserta la ficha
                 if (buscaMejorOpcion(i,j)==importancia){
                    insertarFicha(i,j);
                    insertada=TRUE;
                 }
             }
         }
     }
     return insertada;
}


//-----------------------------------------------------------
//   void copiaArray(BYTE[][] array)
//
//   copia el array leerArray en copiarArray
//
//
//-----------------------------------------------------------
void   copiarArray(BYTE leerArray[MAXFILAS][MAXCOLS],BYTE copiarArray
[MAXFILAS][MAXCOLS]){


    // recorro el tablero
     WORD i,j;
     for (i=0;i < MAXFILAS;i++){
         for (j=0;j < MAXCOLS;j++){
             copiarArray[i][j]=leerArray[i][j];
         }
     }
     return ;
}
```

```
//------------------------------------------------------------
//   WORD jugadaAutomatica (void)
//
//
//   retorna la importancia con la que se colocó la ficha

//------------------------------------------------------------
 WORD jugadaAutomatica (void){
      BOOL colocada=FALSE;
      WORD importancias [NUMIMPORTANCIAS];
      WORD j=0;
      importancias[0]=GANA;// el jugador gana
      importancias[1]=EVITAPERDER;// el ordenador evita que gane el
otro
      importancias[2]=SIPUEDEGANARESQUINA;// el ordenador puede ganar
la proxima
      importancias[3]=SIPUEDEGANAR;// el ordenador no puede ganar en
la proxima
      importancias[4]=NOPUEDEGANARESQUINA;// el ordenador no puede
ganar en la proxima
      importancias[5]=NOPUEDEGANAR;// el ordenador no puede ganar en
la proxima
      // intenta colocar fichas desde la maxima importancia GANA a la
ultima
      // NOPUEDEGANAR. Cuando logra colocar alguna ficha con la
importancia
      // determinada se para
      while ((!colocada) && (j < NUMIMPORTANCIAS)){
            colocada = colocarFicha(importancias[j]);
            j++;
      }
      --j;
      return importancias[j];
}
```

```
//-----------------------------------------------------------
//     void jugadaAutomaticaDesplaza(void)
//
//    Jugada de la maquina cuando tiene que desplazar
//    Se juega probando desplazar cada ficha de la maquina.
//    se almacena la mejor importancia
//    y finalmente se restaura el tablero con mejor resultado
//    retorna la importancia de la jugada. Esto sirve para que
//    la maquina pueda comprobar si el jugador en el siguiente
//    turno puede ganar la partida por haber desplazado una ficha
//    que evitaba perder.
//
//   nota ** dentro del codigo = comprobar que aunque es una buena
//   jugada evita que al
//   haber desplazado la ficha, el nuevo hueco libre sirva
//   para que gane el jugador.
//   retorna importancia de la jugada
//-----------------------------------------------------------

void jugadaAutomaticaDesplaza(void){
     WORD mejorImportancia; // mejor importancia conseguida hasta el
momento
     WORD importanciaNueva;// importancia de la ultima jugada
     WORD temp;
     BYTE
tempTablero[MAXFILAS][MAXCOLS],mejorTablero[MAXFILAS][MAXCOLS];
     BYTE tempNumFichas=numeroFichas;
     WORD i,j;
     copiarArray(tablero,tempTablero); //hacer copia del tablero
     copiarArray(tablero,mejorTablero); // inicializar mejortablero
con actual
     // recorrer el tablero
     mejorImportancia=-1;
     importanciaNueva=-1;
     for (i=0;i < MAXFILAS;i++){
         for (j=0;j < MAXCOLS;j++){
             copiarArray(tempTablero,tablero);// cada vez jugamos con
las posiciones iniciales del turno
             numeroFichas=tempNumFichas;
             // probar jugada en todos las celdas donde hay fichas de
la maquina
             if (tablero[i][j] ==simbolo(turnoActual)) {
                // almacenar la ficha que se quita para falsos
EVITAPERDER
                // (la maquina no moveria en esta caso si no puede
ganar)

                if (FichaSeMueve(i,j)==TRUE) {
                    quitarFicha(i,j);
                    importanciaNueva = jugadaAutomatica();
```

```
                       if
(esMejorJugada(mejorImportancia,importanciaNueva)){
                           // ver nota ** en inicio. Comprobar que
juega la maquina
                           temp=jugadaContrario();
                           if (((temp!=GANA)) ||
(importanciaNueva==GANA)){
                              // el jugador no puede ganar en la
proxima jugada.
                              // guardamos el tablero con la mejor
jugada hasta el momento
                              copiarArray(tablero,mejorTablero);
                              mejorImportancia = importanciaNueva;
                           }
                       }
                 }

             }
         }
     }
    // despues de jugar todas las posibilidades restauramos el mejor
tablero

    copiarArray(mejorTablero,tablero);

}

// compureba que la ficha se mueve
BOOL FichaSeMueve (int fila,int  columna){
     BYTE tempTablero[MAXFILAS][MAXCOLS];
     WORD importanciaNueva;
     copiarArray(tablero,tempTablero); //hacer copia del tablero
     quitarFicha(fila,columna);
     importanciaNueva = jugadaAutomatica();
     // comprobar que se ha insertado en la mismo lugar
     if (tablero[fila][columna] ==simbolo(turnoActual)){
      copiarArray(tempTablero,tablero);// restaurar tablero
       return FALSE;
     } else {
         copiarArray(tempTablero,tablero);// restaurar tablero
          return TRUE;
     }
}
```

```
//-----------------------------------------------------------
//   WORD jugadaContrario(void)
//
//
//
//-----------------------------------------------------------

WORD jugadaContrario(void){
     WORD mejorImportancia=-1; // mejor importancia conseguida hasta
el momento
     WORD importanciaNueva=-1; // importancia de la ultima jugada
     BYTE
tempTablero[MAXFILAS][MAXCOLS],mejorTablero[MAXFILAS][MAXCOLS];
     BYTE tempNumFichas=numeroFichas;
     WORD i,j;
     copiarArray(tablero,tempTablero); //hacer copia del tablero
     cambiaJugador(); //jugar simulando ser el jugador para ver si
gana
         // recorrer el tablero

     for (i=0;i < MAXFILAS;i++){
         for (j=0;j < MAXCOLS;j++){
             copiarArray(tempTablero,tablero);// cada vez jugamos con
las posiciones iniciales del turno
             numeroFichas=tempNumFichas;
             // probar jugada en todos las celdas donde hay fichas de
la maquina
             if (tablero[i][j] ==simbolo(turnoActual)) {
                quitarFicha(i,j);
                importanciaNueva = jugadaAutomatica();
                if
(esMejorJugada(mejorImportancia,importanciaNueva)){
                            copiarArray(tablero,mejorTablero);
                            mejorImportancia = importanciaNueva;
                }
             }
         }
     }
     cambiaJugador(); //jugar simulando ser el jugador para ver si
gana
     copiarArray(tempTablero,tablero); //restaurar copia del tablero
     return mejorImportancia;
}
```

```
//----------------------------------------------------------
// void ajustaTimer1(void)
//
// Inicializa la interrupcion del Timer1. y arranca el timer1
//
//
//----------------------------------------------------------
void ajustaTimer1(void){

     // ajustar TMR1 para 1 ms RST=0
     mbar_writeShort(MCFSIM_TMR1,TMR1_CONF);
     // poner a 0 el valor de la cuenta TCN0
     mbar_writeShort(MCFSIM_TCN1,0); //cualquier valor pone a 0
     // ajustar valor comparacion de la cuenta
     mbar_writeShort(MCFSIM_TRR1,TRR1_CONF);


     // borrar REF en TER1 (tambien borra PI en ICR1)
     mbar_writeShort(MCFSIM_TER1,3); //poner a 0 cap y ref
(escribiendo 1)
     // arrancar TMR1 RST=1
     mbar_writeShort(MCFSIM_TMR1,TMR1_ON);

     // configurar interrupciones timer1 en ICR1
     mbar_writeLong(MCFSIM_ICR1, 0x88888B88); // TIMER1 IPL=3 Y PI=1
(PARA CONSEGUIR PI=0)


     // habilitar interrupciones
     sti();
}
```

```
//-----------------------------------------------------------
// void ajustaTimer0(WORD ttr)
//
//
//   Reprogama el timer para reproducir una nueva nota
//   el timer0 NO INTERRUMPE!!!!
//
//-----------------------------------------------------------
void ajustaTimer0(WORD trr){

     // ajustar nuevo reg de referenica
     if (trr==0){
        // nota de frec=0 Parar timer
        mbar_writeShort(MCFSIM_TMR0,TMR0_OFF);
     }else {
        // parar hasta reajuste del timer y configurarlo
        mbar_writeShort(MCFSIM_TMR0,TMR0_OFF);
        // poner a 0 el valor de la cuenta TCN0
        mbar_writeShort(MCFSIM_TCN0,0); //cualquier valor pone a 0
        // ajustar nuevo TRR
        mbar_writeShort(MCFSIM_TRR0,trr);
        // arrancar timer
        mbar_writeShort(MCFSIM_TMR0,TMR0_ON);
     }
}



//-----------------------------------------------------------
// WORD calculaPeriodo(WORD frecuenciaNota)
//
//
//   Sirve para cacuclar el nuevo periodo del timer0 para
//   generar la onda cuadrada.
//   frecuenciaNota = frecuencia del tono de la nota
//   Retorna el valor de comparacion del timer (TRR0). T=1/(2f)
//
//-----------------------------------------------------------
WORD calculaPeriodo(WORD frecuenciaNota){
     LONG temp;
     if (frecuenciaNota==0){
        return 0; //hay que parar timer
     }else {
        // [(66*10^6)/(2*16*48)]= 66*10^6/1536 = FACTOR_TMR0
      temp=(LONG)FACTOR_TMR0*(LONG)frecuenciaNota;
      temp=MCF_CLK/temp;
        return (WORD)temp;
     }
}
```

```
//-----------------------------------------------------------
// void toutMelodia(void)
//
//
//   Gestiona la melodia de fondo
//   Si debe cambiar de nota llama a cambianota
//-----------------------------------------------------------
void toutMelodia(void){

  if (duracionNotaActual>tiempoMelodia[duracionTotalNotaActual]){
         cambiaNota();
     }

}




//-----------------------------------------------------------
// void toutPerdedor(void)
//
//
//   Gestiona la melodia del perdedor
//   Si debe cambiar de nota o pararse llama a cambiaNotaPerdedor
//-----------------------------------------------------------
void toutPerdedor(void){

  if
(duracionNotaActual>timeMelodiaPerdedor[duracionTotalNotaActual]){
         cambiaNotaPerdedor();
     }

}



//-----------------------------------------------------------
// void toutGanador(void)
//
//
//   Gestiona la melodia del ganador
//   Si debe cambiar de nota o pararse llama a cambiaNotaGanador
//-----------------------------------------------------------
void toutGanador(void){

  if
(duracionNotaActual>timeMelodiaGanador[duracionTotalNotaActual]){
         cambiaNotaGanador();
     }

}
```

```
//-----------------------------------------------------------
// void cambiaNotaGanador(void)
//
//
//   cambia una nota de la melodia de Ganador
//
//-----------------------------------------------------------
void cambiaNotaGanador(void){
       notaActual++;
         duracionTotalNotaActual++;
         duracionNotaActual=0;
         if (notaActual>MAX_TONOS_GANADOR-1){
            //Parar la melodia
            suenaGanador=FALSE;
            ajustaTimer0(0);
         } else {
            ajustaTimer0(calculaPeriodo(melodiaGanador[notaActual]));
         }
}



//-----------------------------------------------------------
// void cambiaNotaPerdedor(void)
//
//
//   cambia una nota de la melodia de Ganador
//
//-----------------------------------------------------------
void cambiaNotaPerdedor(void){
             notaActual++;
         duracionTotalNotaActual++;
         duracionNotaActual=0;
         if (notaActual>MAX_TONOS_PERDEDOR-1){
            //Parar la melodia
            suenaPerdedor=FALSE;
            ajustaTimer0(0);
         } else {

ajustaTimer0(calculaPeriodo(melodiaPerdedor[notaActual]));
         }
}
```

```
//-----------------------------------------------------------
// void cambiaNota(void)
//
//
//   cambia una nota de la melodia de fondo
//
//-----------------------------------------------------------
void cambiaNota(void){
       notaActual++;
         duracionTotalNotaActual++;
         duracionNotaActual=0;
         if (notaActual>MAX_TONOS_MELODIA-1){
            //volver al inicio de la melodia
            notaActual=0;
            duracionTotalNotaActual=0;

         }
         // reprogramar timer con la nueva tecla
          ajustaTimer0(calculaPeriodo(melodia[notaActual]));

}




//-----------------------------------------------------------
// void __init(void)
//
//   Funcion por defecto de inicialización del sistema
//
//-----------------------------------------------------------
void __init(void)
{

    // iniciar udp
   iniciarUdp();
    // iniciar escucha mensajes 3r
    EscuchaUdp(puertoEscucha3R,(long) &recibeUdp3R);
    // iniciar escucha mensajesdebugger
    EscuchaUdp(puertoEscuchaDEBUG,(long)&recibeUdpDebugger);

}
```

```
//-----------------------------------------------------------
// Definicion de rutinas de atencion a la interrupción
// Es necesario definirlas aunque estan vacias
void rutina_int1(void){}
void rutina_int2(void){}
void rutina_int3(void){}
void rutina_int4(void){}
void rutina_tout0(void){}



void rutina_tout1(void){
     tiempoTurno++;
     duracionNotaActual++;

     // comprobar tout jugada
     if (tiempoTurno>MAX_TURNO){
       // no se puede perder el turno si el primer jugador no ha
colocado en el centro
       if (numeroFichas>1){
        finTurno=TRUE;
       }
     }

      // relanzar melodia fondo despues de partida
     if ((suenaGanador==FALSE) & (suenaPerdedor==FALSE) &
(suenaMelodia==FALSE)){
          iniciaMelodia();
     }
     // actualizar melodia
     if (suenaMelodia==TRUE) toutMelodia();
     if (suenaPerdedor==TRUE) toutPerdedor();
     if (suenaGanador==TRUE) toutGanador();
     // borrar REF en TER1 (tambien borra PI en ICR1)
     mbar_writeShort(MCFSIM_TER1,3); //poner a 0 cap y ref
(escribiendo 1)

return;
     }
void rutina_tout2(void){}
void rutina_tout3(void){}
```

```
void rutina_ethernetrx(void){
  fec_handler(&fec_nif);
}
void rutina_ethernettx(void){}
```

10.4 CODIGO APLICACIÓN 3 EN RAYA (COLDFIRE) 3RAYA.LIB

```
/*   CLIENTE UDP DESARROLLADO POR
        FERNANDO NUÑEZ SERRANO
        ALEJANDRO BADOLATO MARTIN
        ver. 28-05-2008 10:53 / Codigo definitivo

     Cuando se recibe un mensaje UDP en el puerto de
     escucha se le añade ## al final de la cadena y se
     introduce en bufferUdpRX[]. Además se pone a true
     el flag mensajeUdpRx para indicar la llegada del
     mensaje. El mensaje no debe ser superior a MAXMENSAJEUDP

     Para enviar mensajes se deben recibir cadenas de texto
     con terminación ## para que pueda ser detectado el fin de
     la cadena por parte de la rutina longitud
*/

#include "stdlib.h"
#include "m5272c3.h"
#include "nif.h"
#include "fec.h"
#include "arp.h"
#include "ip.h"
#include "udp.h"
#include "tftp.h"
#include "timer.h"
#include "tftp_main.h"
#include "m5272lib.h"
#include "mcf5xxx.h"

#include "m5272lib.c"
#include "stdlib.c"
#include "timer.c"
#include "fec.c"
#include "arp.c"
#include "ip.c"

extern BOOL mensajeUdpRx;

#define puertoDestinoDEBUG 5000 // puerto destino para envio UDP
DEBUGGER UDP
#define V_BASE 0x40            // Dirección de inicio de los vectores
de interrupción
#define DIR_VETHRX 4 * (V_BASE + 22)     // Dirección del vector de
interrupcion de ethernetrx
#define DIR_VETHTX 4 * (V_BASE + 23)     // Dirección del vector de
interrupcion de ethernettx
```

```
#define V_BASE 0x40                  // Dirección de inicio de los
vectores de interrupción
#define DIR_VTMR1 4*(V_BASE+6)             // Dirección del vector de
TMR1
#define MAXMENSAJEUDP 320 // maximo número de caracteres a introducir
en buffer RX
#define MAXBLOQUEMEMORIA 320 // maximo bloque de memoria que se puede
enviar para depuracion (bytes)

#define TIMER_RED 2// TIMER PARA LA RED

/*******************************************************************
/
/* Definir interface del FEC */
NIF       fec_nif;
IP_INFO        ip_info;
ARP_INFO  arp_info;

//RSSH: #define USERSPACE (SDRAM_ADDRESS + 0x40000)
#define USERSPACE (SDRAM_ADDRESS + 0x0000)

static IP_ADDR cliente = {192,168,1,20}; //coldfire
static IP_ADDR gateway = {192,168,1,1};
static IP_ADDR netmask = {255,255,255,0};
static IP_ADDR servidor = {192,168,1,36}; //pc
static unsigned char mac[6] = {0x00, 0x0B, 0xCB, 0xFF, 0xF0, 0x00};
static int puertoOrigen = 4000; // puerto de origen para envio UDP

static int offsetUdp= UDP_HDR_OFFSET+8; // offset para datos trama
UDP

NBUF *pNbuf;
char bufferUdpRX[MAXMENSAJEUDP];//buffer de recepción
char *pbufferUdpTX;// puntero al buffer de tx
char *pbufferUdpRX;// puntero al buffer de rx




/*******************************************************************
/

void recibeUdp (NIF *nif, NBUF *pNbuf, int hdr_offset);
int longitud(char* cadena);
void enviaUdp (char* mensaje, int puertodest);
void interrumpe (void);
long extraerDireccion (char* mensaje);
void accedeMemoria(char* puntero, char tip);
long extraerLong (char* mensaje);
```

```
// inicializa el cliente UDP
void iniciarUdp (void){
        mbar_writeByte(MCFSIM_PIVR,V_BASE);
        ACCESO_A_MEMORIA_LONG(DIR_VTMR1)= (ULONG)_prep_TOUT1; //
Escribimos la dirección de la función para TMR0
        ACCESO_A_MEMORIA_LONG(DIR_VETHRX) = (ULONG) _prep_ETHRX;
     // Escribimos la dirección de la función de atención a la
interrupción EthernetTx
        ACCESO_A_MEMORIA_LONG(DIR_VETHTX) = (ULONG) _prep_ETHTX;
        // Marcamos la interrupción EthernetRx de nivel 6
        mbar_writeLong(MCFSIM_ICR3, 0x88888E88);
        // Marcamos la interrupción EthernetTx de nivel 6
        mbar_writeLong(MCFSIM_ICR1, 0x888888E8);
        // Inicializar puntero del los buffer UDP
        pbufferUdpRX=&bufferUdpRX[0];
        // Inicializar timer para el uso de la red
     timer_init(TIMER_RED, TIMER_NETWORK_PERIOD,
                SYSTEM_CLOCK, TIMER_NETWORK_LEVEL);
     /* Activar las interrupciones del FEC  */
     mbar_writeLong(MCFSIM_ICR3,MCF5272_SIM_ICR_ERX_IL(FEC_LEVEL));
     /* Inicializar el dispositivo de red */
     fec_init(&fec_nif);
     /* Poner la direccion MAC en NIF */
     fec_nif.hwa[0] = mac[0];
     fec_nif.hwa[1] = mac[1];
     fec_nif.hwa[2] = mac[2];
     fec_nif.hwa[3] = mac[3];
     fec_nif.hwa[4] = mac[4];
     fec_nif.hwa[5] = mac[5];
     /* Inicializar buffers de red */
     nbuf_init();
     /* Inicializar ARP */
     arp_init(&arp_info);
     nif_bind_protocol(&fec_nif,FRAME_ARP,(void *)arp_handler,(void
*)&arp_info);
     /* Inicializar IP */
     ip_init(&ip_info,cliente,gateway,netmask);
     nif_bind_protocol(&fec_nif,FRAME_IP,(void *)ip_handler,(void
*)&ip_info);
     /* Inicializar UDP */
     udp_init();
        /* Resetear el controlador Ethernet */
     fec_nif.reset(&fec_nif);
     fec_nif.start(&fec_nif);
     /* Utilizar la direccion mac para generar un puerto pseudo
aleatorio */
     udp_prime_port((uint16)((fec_nif.hwa[4] << 8) |
fec_nif.hwa[5]));
}
```

```
void EscuchaUdp (int puerto,long rutina){
        // Escuchar en puerto
        // rutina apunta a la direccion de la rutina de atencion
        udp_bind_port(puerto, rutina);
}




void recibeUdp3R (NIF *nif, NBUF *pNbuf, int hdr_offset){

      int finBucle,i; // fin del bucle extraccion datos
      char *pchar; // puntero al inicio de la cadena

        udp_frame_hdr *udpframe;

     /* accedieno a la memoria compartida, deshabilitar
interrupciones */
     asm_set_ipl(6);

        // extraer trama UDP de pNbuf
     udpframe = (udp_frame_hdr *)&pNbuf->data[hdr_offset];
        pchar=(char*)udpframe;

        // ajustar fin del bucle
        if ((pNbuf->length)>MAXMENSAJEUDP ){
               finBucle=MAXMENSAJEUDP ;
        }else finBucle=pNbuf->length;

        // bucle extraccion de datos
        for(i=0; i<finBucle; i++){
           bufferUdpRX[i] =pchar[i];
        }
         // añadir fin de mensaje por si no lo tenia
          i++;
          bufferUdpRX[i]='#';
          i++;
          bufferUdpRX[i]='#';
         // marcar flag mensaje recibido
         mensajeUdpRx=TRUE;
         asm_set_ipl(0);
}
```

```
void recibeUdpDebugger (NIF *nif, NBUF *pNbuf, int hdr_offset){
     char *pchar; // puntero al inicio de la cadena
     int finBucle; // fin del bucle extraccion datos

        udp_frame_hdr *udpframe;
        /* accedieno a la memoria compartida, deshabilitar
interrupciones */
     asm_set_ipl(6);
         // extraer trama UDP de pNbuf
     udpframe = (udp_frame_hdr *)&pNbuf->data[hdr_offset];
        pchar=(char*)udpframe;
        // ajustar fin del bucle
        if ((pNbuf->length)>MAXMENSAJEUDP ){
               finBucle=MAXMENSAJEUDP ;
        }else finBucle=pNbuf->length;
        if (pchar[0]=='1') {
            //solicitidada variable
        }
        switch (pchar[0]){
              case '1':
                  // solicitada variable
                  accedeMemoria(pchar,'1');
                   break;
              case '2':
                 // solicitado bloque memoria
                  accedeMemoria(pchar,'2');
              break;
              case 'A':
                 // solicitado interrumpir programa
                 interrumpe();
              break;
        }
}
```

```
// accede a memoria y envia los bytes seleccionados

void accedeMemoria(char* puntero, char tip){
        long direccion; // direccion de memoria a la que se quiere
acceder
     long totalBytes; // numero total de bytes a leer
        int i;
     char dato[4+MAXBLOQUEMEMORIA]; // datos leidos de la memoria = 4
cabeceras + MAXBLOQUEMEMORIA

       // borrar buffer envio
         for (i=0;i<5+MAXBLOQUEMEMORIA ;i++){
              dato[i]='0';
         }
         // poner cabecera envio variable = 1#dato
         dato[0]=tip;
         dato[1]='#';
        // extraer direccion
        direccion =  extraerLong (puntero+2);
        // extraer numero total de bytes a leer
        totalBytes=extraerLong(puntero+11);
        for (i=0;i<totalBytes;i++){
               dato[i+2]=lee_byte(direccion+i);
         }
         i=i+2;
         dato[i++]='#';
         dato[i]='#';
         asm_set_ipl(0);
         enviaUdp (&dato[0],puertoDestinoDEBUG);

}
```

```
// poner un breakponit para poder interrumpir
void interrumpe (void){
      output("Habilite la interrupcion\r\n"); // poner break a esta
linea
}
```

```
// extrae la un long del mensaje ASCII
long extraerLong (char* mensaje){
          long dir;
          long valor;
          int i,j;
       long factor;

             // inicializar direccion
              dir=0;
             for (i=0;i<8;i++){
                   valor=(int)mensaje[7-i];
                   factor=1;
             for (j=0;j<i;j++){
                     factor=factor*16;
                   }

                   if ((valor>=48) & (valor<=57)) {
                         // es un numero del 0 al 9
                         valor=valor-48;
                         dir+=valor*factor;
                   } else{
                        // if ((valor>=41) & (valor<=46)) {
                     // es una letra de la A la F
                valor=valor-55;
                         dir+=valor*factor;
                        // }
                   }

          }
     return dir;
}
```

```
// puerto=puerto Destino
void enviaUdp (char* mensaje, int puertodest){

     char *pchar; // puntero al inicio de la cadena
        int i;
        pchar=mensaje;
       /* Deshabilitar interrupciones */
     asm_set_ipl(6);
     pNbuf = fec_nif.tx_alloc();

     /* Establecer la longitud del paquete UDP*/
     pNbuf->length = longitud(mensaje);

       // Establecer mensaje
        //dirbasura=&mensaje[i];
        for(i=0; i<pNbuf->length; i++){
              pNbuf->data[offsetUdp+i]=mensaje[i];
          }

        /* Enviar el paquete */
     udp_send(&fec_nif, servidor, puertoOrigen, puertodest, pNbuf);
     /* Habilitar las interrupciones para poder recibir UDP */
     asm_set_ipl(0);

}
```

```
// puntero apunta al inicio del array de char
// retorna longitud de la cadena. Maxima longitud
// permitida = MAXMENSAJERX
int longitud(char* cadena){

BOOL fin;
int contador;
fin=FALSE;
contador=0;
  while(fin==FALSE){
      if (cadena[contador]=='#'){
          if (cadena[contador+1]=='#'){
                fin=TRUE;
          }
       }
      if (contador>MAXMENSAJEUDP ) fin=TRUE;
      contador++;
  }
return --contador;
}
```